Liberating Virtual Machines from Physical Boundaries through Execution Knowledge

Yoshihisa Abe

CMU-CS-15-147

December 2015

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan, Chair
Dan Siewiorek
Todd Mowry
Kaustubh Joshi, AT&T Research
Roxana Geambasu, Columbia University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported by the National Science Foundation under grant numbers CNS-0833882, CCF-1019104, IIS-1065336, and CNS-1518865; by the Defense Advanced Research Projects Agency under grant numbers FA8721-05-C-0003 and FA8650-11-C-7190; by the Moore Foundation under grant number 2160; by the Alfred P. Sloan Foundation under grant number 20121031; by an Intel Science and Technology Center grant; by the Quality of Life Technologies ERC (NSF-EEC-0540865); and by the Department of Defense under Contract No. FA8721-05-C-0003 for the operation of the Software Engineering Institute, a federally funded research and development center. Additional support was provided by Intel, IBM, Google, Bosch, Vodafone, Crown Castle, and the Conklin Kistler family fund. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Virtualization, Migration, Operating systems, Cloud computing, Interactivity

Abstract

Hardware virtualization enables remote instantiation of computation through the preserved executability of encapsulated software. The large size of virtual machines (VMs), however, poses challenges in exploiting this strong feature under the existence of resource constraints. In this thesis, we claim that the use of execution knowledge achieves the efficiency and timeliness of VM state transfer in such environments. We demonstrate its effectiveness in two concrete contexts in which the challenges materialize: 1) VM delivery over WANs, with network resource limitations, and 2) urgent migration of VMs under contention, with strict time requirements. In the context of VM delivery over WANs, we take advantage of the knowledge about past VM execution instances. We conduct the evaluation of `vTube`, a system for efficiently streaming virtual appliances, from both systems and human-centric perspectives. In the context of urgent migration of VMs under contention, we leverage current execution knowledge at the guest OS level. Our approach, called enlightened post-copy, uses this knowledge to expedite the resolution of contention between VMs. Our proposed solutions address the corresponding problems by providing VM performance as defined by critical metrics in their specific contexts.

Acknowledgements

I am truly thankful to my adviser, Prof. Mahadev Satyanarayanan, and my internal committee members, Prof. Dan Siewiorek and Prof. Todd Mowry, for guiding me with invaluable feedback and insights. I am also very grateful to my external committee members and long-term collaborators, Dr. Kaustubh Joshi and Prof. Roxana Geambasu, for their continued support and extensive contributions. The work on interactivity evaluation of VM systems would not have been feasible without our collaboration with Brandon Taylor. The past and current members of the Elijah Group, especially Jan Harkes, Benjamin Gilbert, Wolf Richter, and Kiryong Ha, have offered many opportunities for collaboration and productive discussions.

Completing this thesis was made possible by Deborah Cavlovich and Catherine Copetas, who always provide great assistance to the members of the Computer Science Department.

Last but not least, I am indebted to my parents for their invariable supportiveness throughout my graduate education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hardware virtualization[1] is a technology that exposes virtual machines (VMs), a layer of emulated hardware on top of physical hardware, to software. The domain of software execution inside VMs is called *guest*, whereas that managing real hardware is referred to as *host*. Virtualization is achieved through software emulation of devices, often with native hardware support for improving the speed of guest execution. The systems software that creates the layer of virtualized hardware is called *hypervisor*. Hypervisor cleanly separates guests from physical hardware, creating the illusion that they are running directly on a physical machine.

A key benefit of virtualization is guaranteed executability of software. The guest is able to run regardless of the hardware and software environment of the physical machine. Therefore, software needs to be prepared once for each virtual machine, rather than for each machine on which it may run. Furthermore, this executability provides the capability to transfer computation. Pre-configured VM images can be copied to a remote machine for instantiation, or running VMs can be transferred to a remote machine for continued execution on the remote site. Unfortunately, the applicability of these operations is largely limited by the VM state size. In order to transfer a VM from one machine to another, the state for its memory, disk, and peripheral devices must be transferred. Some of these states are often prohibitively large; in particular, memory can be several gigabytes or more, and disks in the order of tens of gigabytes. Even with the network speeds improving over the past decades, naively transferring this much state takes a significant amount of time and incurs performance degradation. Moreover, the need for fast VM transfer is magnified under limited resource availability, which requires efficient solutions for meeting performance requirements.

[1] In the rest of this thesis, we will refer to hardware virtualization simply as virtualization.

Table 1.1 summarizes contexts that demand fast VM transfer, grouped into two categories: VM delivery over wide-area networks (WANs) and urgent transfer of VMs under contention. They are characterized by specific use cases, primary metrics of importance, and the forms of VM transfer. VM delivery over WANs is a key mechanism for distributing executable contents to end users, whose native computing environments are outside the control of their distributors. In particular, when they contain legacy applications, virtualization is a prerequisite for ensuring their executability. VMs can also be used to cleanly package applications and their particular dependencies, such as codecs for uncommon or proprietary media formats. This packaging avoids affecting the user's host environment and eliminates the burden of manual installation and management. VM delivery also provides a convenient way of moving computation to the user's large local data. Over WANs, available bandwidth is limited to tens of megabytes per second or often much less. Also, round-trip latency can reach as high as hundreds of milliseconds. This limited availability of network resources poses a fundamental challenge for timely remote instantiation of VMs.

In clouds, fast VM transfer between machines can be used to cope with dynamically changing loads that exceed the capacity of physical resources. It mitigates the tension between resource efficiency and performance preservation. On one hand, efficient use of physical resources reduces hardware cost. On the other, running multiple VMs on the same machine for resource efficiency sacrifices their performance under unexpectedly high loads. Fast VM transfer allows reactive re-allocation of VMs to machines in such situations, resolving resource contention. Additionally, such contention can include adverse scenarios. For example, a denial-of-service (DoS) attack maliciously floods a service with unsolicited requests. A VM under this type of attack may render other VMs on the same host unresponsive. VM re-allocation under high loads can also be beneficial in more distributed environments, such as Cloudlets [104]. Although good network connectivity is expected in these use cases, they have high requirements for VM performance. Resolving contention requires that transfer be performed as fast as possible, as otherwise the hosted services continue to be disrupted.

In the above contexts, existing approaches to VM transfer are not sufficient for satisfying their requirements. Over WANs, current solutions are predominantly in the form of VM image downloading. Without the existence of non-trivial infrastructures such as VM distribution networks, clients need to wait for an extended period of time before launching a VM. In cloud settings, various forms of transfer have been studied and incorporated into hypervisor implementations. However, these transfer methods typically focus on specific aspects of performance, and are ineffective when dealing with highly loaded VMs. For example, *live migration* [40, 90] minimizes down time, during which VM execution is

Table 1.1: Challenging Contexts in Need of Fast VM State Transfer

Contexts	VM delivery over WANs	Urgent transfer of VMs under Contention
Use Cases	VMs for end users: - Archival VMs - Aapplication/media content packaging - Computation transfer to site of data	VMs in clouds: - Overprovisioning - Segregation of resource-hogging VMs
Primary Metrics	1. Time to VM instantiation 2. VM user experience	1. Execution transfer time 2. Application service quality 3. Total duration
Transfer Targets	Static VM images	Running VM instances
Execution Knowledge	Past execution records at hypervisor level	Current execution status at guest OS level
Solution	vTube (Chapter 3)	Enlightened Post-Copy (Chapter 5)

suspended, at the cost of delayed execution transfer. The underlying problem common to VM downloading and existing transfer methods is the lack of understanding and adapting to VM behavior. This results in the coarse granularity of these approaches that suffices for either timeliness or performance, but not both as required in our target contexts.

In this thesis, we address the current limitations of VM transfer by exploiting knowledge about VM execution. Performing state transfer using this knowledge makes it feasible to instantiate VMs in a timely manner. In turn, the executability of software can be extended and applied to remote instantiation without severe performance degradation. Our central claim is as follows:

Thesis statement: *Hardware virtualization preserves the executability of software across physical machines, but large VM state size discourages fast remote instantiation of computation. This obstacle can be overcome by state transfer that is guided by knowledge from past and present execution instances. It achieves reduced launch delay and runtime overhead in bandwidth-challenged and resource-overloaded settings.*

We validate our claim by showing that informed state transfer enables fast VM launch in the contexts in Table 1.1, exploiting appropriate execution knowledge and adequately providing performance as defined by context-specific metrics. For VM delivery over WANs, the use cases are centered around the distribution of *virtual appliances*, which are VM images pre-configured for specific applications. Taking advantage of this property, we exploit past execution as the source of knowledge. We analyze the previous

execution instances of a given virtual appliance, and derive its VM state access patterns. These access patterns are used to estimate the state required by the currently execution instance in the near future. VM state is delivered to the client according to this estimation. The primary performance metrics in this context are time to VM instantiation on the client side, and VM user experience including the wait time for instantiation and VM performance afterwards. We develop and evaluate vTube, a system for streaming VMs over low-bandwidth, high-latency networks. We also conduct a series of user studies for aspects of VM performance that elude evaluation from the systems perspective. For VMs in clouds, we propose a new approach to VM state transfer, called enlightened post-copy. Enlightened post-copy targets running VM instances, and exploits the guest's knowledge about current execution for high efficiency. This knowledge is used to transfer VM state to the destination in a prioritized manner. The main metrics of interest in this context are time to transfer VM execution, the service quality of applications in the VM, and the total duration of the operation. vTube and enlightened post-copy each apply the concept of state transfer with execution knowledge to the corresponding use cases. With these approaches, we significantly expand the range of contexts to which VMs offer a viable solution.

Figure 1.1 shows the outline of this thesis work. The numbers at the lower right corner of boxes represent the chapter that covers the respective topic. Our thesis statement develops into the two distinct and complementary contexts. We apply past execution knowledge per virtual appliance to the problem of VM delivery over WANs, proposing `vTube` as our solution. We perform experiments to evaluate its streaming performance on real-world networks, including 3G/4G LTE networks and public Wi-Fi's. This work is presented in Chapter 3. Next, we assess the system usability of `vTube` as perceived by human subjects, which augments the systems-level results covered in Chapter 3. We also take a common alternative approach to the VM streaming model of `vTube`, remote VM access using a popular implementation called Virtual Network Computing (VNC) [98], and compare the trade-offs of the two systems. Chapter 4 describes a series of user studies for this evaluation. In the other context, VMs under resource contention, we exploit current execution knowledge and propose enlightened post-copy. We evaluate its performance against existing migration approaches, including live migration and baselines such as stop-and-copy and simple post-copy. The work on enlightened post-copy is explained in Chapter 5. The results from Chapters 3, 4, and 5 collectively validate our claim that VM execution knowledge enables timely VM transfer in challenging environments with resource limitations.

In the next chapter, we begin by describing the relevant features of virtualization and reviewing past work in the related areas.

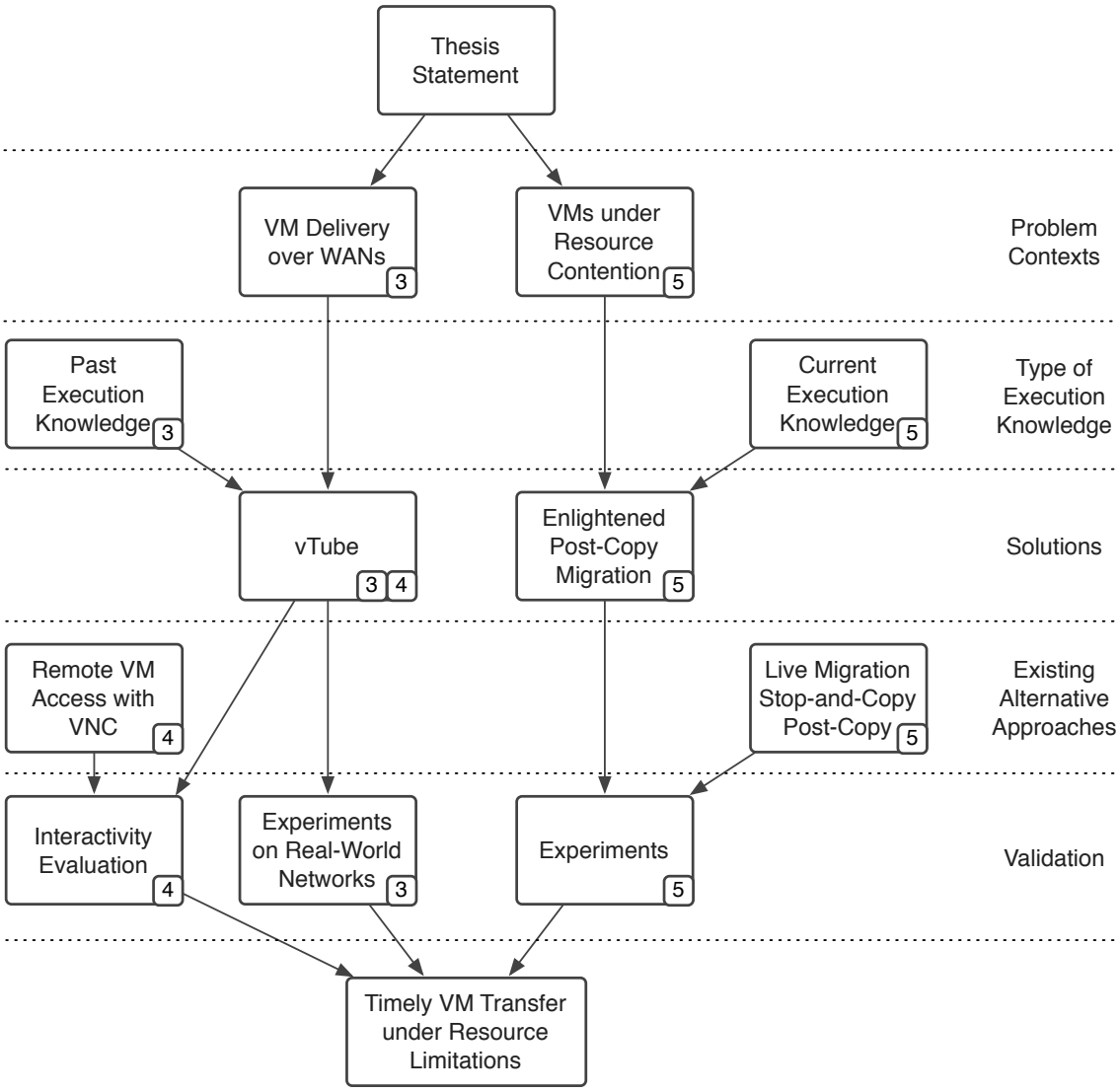


Figure 1.1: Thesis Outline. The numbers represent the chapters that cover the corresponding contents.

Chapter 2

Background

Since its invention, virtualization has evolved into a crucial part of systems that enables software deployment in new ways. The resulting potential is a core requirement in the specific contexts we explore. The first half of this chapter explains how virtualization serves as the basis for our solutions, detailing its strengths of particular importance. After the premises are established, the second half covers related work in virtualization and other areas.

2.1 Essential Properties of Virtualization

Virtualization offers the capabilities to guarantee software executability and instantiate remote computation, achieving both in a robust manner. These fundamental attributes are the grounds on which we develop our work.

2.1.1 Executability Guarantee

Virtualization originates in the era of mainframe computers, such as VM/370 [41]. At the time, it was used for partitioning hardware resources into multiple entities available for software. Disco [34, 49] revitalized virtualization in the late 1990's, introducing its values for multi-processor machines. In this context, the key strength of virtualization is backward hardware compatibility for operating systems; the computational resources of high-end, multi-processor machines can be utilized without the significant development effort for creating a new OS, by means of running multiple instances of VMs each running

an existing commodity OS. This backward compatibility is achieved through separating the physical hardware and the VM's view of hardware, and is an essential property of virtualization that provides executability guarantee for the software inside the VM.

Figure 2.1 illustrates the hardware and software hierarchy in environments with and without virtualization. In non-virtualized environments, the OS executes directly on physical hardware, managing its resources and exporting its abstraction to user applications. In virtualized environments, in contrast, a hypervisor exists between physical hardware and the VM. Depending on the virtualization architecture, the hypervisor acts as the sole entity that manages the physical hardware, or it runs on a separate OS, called host OS, that manages the hardware. For example, qemu-kvm [7], which we mainly use in this work, is a major example of the latter; qemu-kvm runs as a regular process on host Linux. In both cases, the hypervisor exposes virtualized hardware as the hardware configuration of the VM. The guest OS executes on and manages this virtualized hardware. User applications sits at the top of this hierarchy, running on the guest OS.

The hypervisor controls the virtualized hardware seen by the guest OS. Decoupled from the physical hardware, it remains consistent with the requirements and supports of the guest OS even when the native hardware configuration of the machine changes. Moreover, typically the hypervisor is smaller and more maintainable than a full-fledged OS, in terms of the program size; it therefore has good sustainability across different generations of platforms. These features of virtualization lead to the preserved executability of the guest OS and its applications. As far as the VM is able to run, so are the software layers on top despite possible changes in the physical hardware configurations.

This executability guarantee holds on recent platforms in spite of their advanced support for virtualization, which may violate the strict separation of virtualized hardware from physical hardware. Most machines today have hardware support for virtualization, most notably extended page tables (EPTs) for direct memory address translation for the address spaces of the guest OS. It reduces the cost of virtualization and achieves the performance of VM execution that is comparable to native execution in many cases. Such hardware support requires that the guest and host architectures be compatible in order to exploit the performance benefits. Also, para-virtualization is a common technique employed by major hypervisors. As opposed to full virtualization, which exposes a complete image of existing hardware configuration to the guest, it establishes explicit channels that make the guest OS aware of virtualization. Common examples include device drivers for hard disks and network drivers; the guest communicates with these non-transparent virtual devices in a manner that efficiently translates to the corresponding operations for the underlying physical devices. The hypervisor is still able to retain the execution compatibility under the existence of these performance enhancements. It can absorb differences in

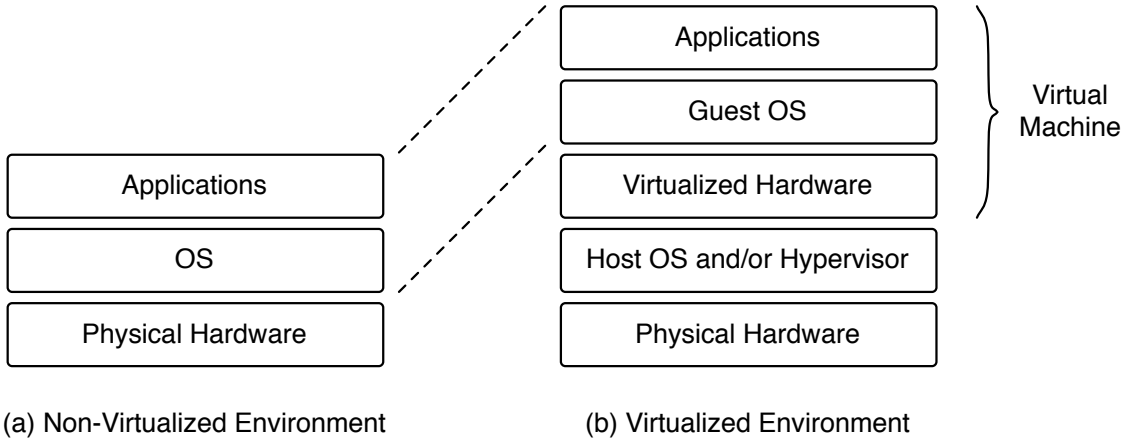


Figure 2.1: Comparison of Virtualized and Non-Virtualized Environments

hardware features by, for example, software emulation, and preserve the compatibility of para-virtualization features by keeping the abstraction exposed to the guest OS intact. The executability guarantee thus remains to be a fundamental property of virtualization despite the advancement of its technologies.

2.1.2 Remote Instantiation of Computation

The executability guarantee of VMs enables them to be moved between physical machines. As long as the hypervisors on these machines provide a consistent virtualized hardware configuration, they can execute the VMs. Remote instantiation of computation encapsulated in VMs, using this property, is the main theme of this thesis work.

VMs in the turned-off state, typically associated with a virtualized device configuration and images of its persistent storage, can be transferred to and booted on an arbitrary machine with an appropriate hypervisor. Also, running VMs can have their state saved on one machine, and then be transferred to and resumed on another machine. This state captures all the entities above the hypervisor level in Figure 2.1. Most notably, the state of running VMs includes their memory image in addition to that of turned-off VMs. When the state of a VM, running or turned off, is saved for use at a later time, it is called a *snapshot*. In the context of VM delivery over WANs presented in Chapter 3, we target virtual appliances maintained in the form of snapshots. When running VMs are relocated to a new machine for continued execution, on the other hand, their state is directly transferred

to the destination machine in stead of being saved as images. *Migration* is a term for this process, and is the type of VM state transfer in the context of urgent VM transfer under contention discussed in Chapter 5. Remote instantiation of computation, in both of these forms, allows the software to move across the boundary between physical machines. We propose approaches to make this capability efficient and applicable in practical situations that suffer from resource constraints.

We achieve such fast instantiation of computation through the use of execution knowledge. The rationale behind this core approach to efficiency is the realization that the state of a VM is inherently tied to its execution. Timely transfer of VM state needs understanding its characteristics, which are determined by the way the computation inside the VM requires it. We capture this relation between computation and VM state as different types of execution knowledge, and apply them in their appropriate contexts. We thus advance the field of VM state transfer, in which existing solutions either treat VM state as simple data or infer its characteristics in an indirect, coarse-grained manner. When we simply download entire VM images, for example, we view them as flat data that do not have particular meanings associated with themselves. As we will explore in the subsequent chapters, more sophisticated approaches in previous work use heuristics that reflect some VM state characteristics; still, they do so without applying the specific concept of execution knowledge, therefore paying the cost of inadequate efficiency in the resource-challenged contexts.

2.1.3 Means of Encapsulation

In addition to regular hardware virtualization, approaches to the encapsulation and transfer of computation have been proposed at various levels in the systems software hierarchy. The early efforts started with process migration, which was supported by systems such as Sprite [46], MOSIX [26], and DEMOS/MP [95]. It can transfer processes between operating systems that execute natively on physical hardware. This approach is direct and efficient, moving the states of processes rather than the guest OS and everything atop as done in virtualization. However, it generally suffers from residual dependencies, in which processes after migration cannot resolve references to resources on the source. Such dependencies include inter-process communication channels and file name spaces. More recently, containers on Linux [109] and jails on FreeBSD [67] provide a mechanism for isolating processes with separate resource access policies. Although lightweight compared to VMs, their security enforcement and migration capability are integrated to a lesser extent. Zap [92] transparently migrates processes between OSes by introducing a lightweight virtualization layer that abstracts the native system resources seen by the processes. In self-migration [55], OSes handle migration by themselves, including their own execution and

thread control. While relying on the L4 microkernel [77] or Xen [29] below the OSes, the responsibilities at the hypervisor level are kept minimal. Unikernels [82] consist of VMs with a single address space, targeting virtual appliances constructed for a specific purpose. Jitsu [81] leverages the unikernel architecture to rapidly deploy applications on distributed, power-efficient devices.

Virtualization, as used in our work, is distinct from these alternatives. First, the guest environment encompasses the sources of residual dependencies, eliminating the complications that exist in process migration. Second, the hypervisor is designed for strong isolation between VMs, preventing one from compromising another on the same host. Third, unlike with the variants of virtualization above, no significant constraints or re-architecting in the guest environment are assumed. These aspects motivate the pursuit of virtualization as the most robust encapsulation of the target software.

2.2 Related Work

The work in this thesis includes multiple research elements, especially efficient VM state transfer, the use of execution knowledge, and evaluation of systems usability. We first review existing solutions to the delivery of VM images and migration of running VMs. We next discuss past work on investigating usability aspects, focusing on the impact of interaction latency, and then different approaches to obtaining execution knowledge and the use of proactive state transfer in other research areas.

2.2.1 VM Delivery and Migration in WAN Environments

Early work on VM transfer started with stop-and-copy, in which guest execution is suspended before, and resumed after, entire state transfer. It was used by Internet Suspend/Resume [71, 106], and adopted by μDenali [117]. Stop-and-copy was also augmented with partial demand fetch and other optimizations [103] for virtualizing the user's desktop environment, as *virtual desktops*, in Internet Suspend/Resume and Collective [38, 101, 102]. Since the advent of these systems, various architectures for distributing VMs across WANs have been proposed. VM distribution networks, such as VDN [94] and VMTorrent [96, 97], disseminate VM images across networked computers to hosts close to the site of VM launch. Huang et al. [61] developed a distributed framework that is aware of the commonality across VM images and distributes them in fine-grained units to cooperating nodes. VMFlock [24] is a framework for efficiently migrating groups of VMs across clouds and data centers. It employs deduplication across VM images to eliminate

redundant state transfer, and state accesses of executing VMs to efficiently retrieve the remaining portions of the images. While our approach to VM delivery shares some of the general techniques common to these distributed architectures, we focus on a server-client model that assumes no wide deployment of intermediate nodes.

VM migration over WANs, for example between data centers, has fostered various optimizations for supporting efficient use of network resources. Bradford et al. [30] addressed two challenges in live migration over WANs, the disruption of services in VMs by throttling their state changes and network redirection by using dynamic DNS reconfiguration. CloudNet [118] provides WAN live migration with optimizations including dynamic iteration control, and deduplication and delta coding of VM state. XvMotion [83] incorporates mechanisms such as efficient disk buffer management and guest throttling to achieve performance comparable to migration in LAN environments. Shrinker [99] applies deduplication and content-based addressing in migrating clusters of VMs across WANs.

2.2.2 VM Migration in Clouds

VM migration in clouds has improved over the past decade, with previous work targeting different environments. In Chapter 5, we will compare our approach, enlightened post-copy, primarily to live migration [40, 90]. Live migration focuses on minimizing the down time of migrated VMs. It exemplifies pre-copy approaches, which have all state transferred to the destination upon the completion of migration. The approach opposite to the pre-copy style is post-copy migration. VMs are resumed on their destination first, and then their state is retrieved. Examples of this approach include work by Hines et al. [56, 57] and by Liu et al. [78]. Post-copy migration is often desirable when migrating all state as fast as possible is prohibitive with respect to available network resources. Hybrid approaches utilizing pre-copy and post-copy have also been proposed [80]. In addition to the transfer of computation, migration in clouds has also been used to provide the elasticity of VM-hosted services. VM cloning, such as Kaleidoscope [33] and SnowFlock [73], allows load balancing by rapidly instantiating VMs to increase the capacity of the hosted services.

Live Migration Enhancements

Being the current standard for migration, live migration has led to various efforts to enhance its performance. As used also in WAN environments, deduplication and compression of VM state have become a fundamental strategy for efficient VM migration. Jo et al. [64, 65] presented an optimization that exploits duplicate contents between memory and persistent storage of the VM. It reduces the duration of migration by skipping the transfer

of such memory contents and having them directly loaded from the storage on the destination. Svärd et al. [111] studied the benefits of applying delta compression to memory pages transferred during live migration. The use of compression allows live migration to scale to heavy workloads in the VM, or to low bandwidth with respect to the workload size, by increasing the speed of state transfer and reducing down time. Hacking and Hudzia [54] also investigated the use of delta compression for migrating large VMs. They also used background proactive state transfer to a potential destination before actual migration is triggered. Delta compression is complementary to our work on enlightened post-copy, and can enhance its performance when the host has spare cycles for the computation.

Working set estimation based on access traces is used for improving the efficiency of live migration [78], while also applied to resume from snapshots [119]. In `vTube`, we convert working set information in traces into fine-grained state access patterns, and dynamically apply it based on the behavior of the running VM.

Scatter-gather live migration [45] addresses the eviction time of VMs under migration for fast resource freeing on their source. By disseminating VM state to intermediate nodes and having the destination receive the state spread over them, it allows evicting the VM state on the source without being constrained by the network capacity of the destination. Our work on enlightened post-copy also has an emphasis on the fast eviction of the target VM from its destination.

In addition to these improvements to the software logic of migration, advancements in data-center hardware have also been contributing to its efficiency. In particular, remote direct memory access (RDMA) improves the speed of state transfer between hosts, thereby enabling migration to handle increasing VM sizes. Huang et al. [62] evaluated an implementation that uses RDMA over InfiniBand, and achieved up to 80% reduction in the duration of migration through eliminating the regular overhead in the software networking stack.

Performance Characterization

One of the challenging problems concerning live migration is quantifying its intricate performance characteristics. Hu et al. [60] developed a framework for automating the measurement of factors such as the down time, total duration, and network transfer amount, and used it with different hypervisors. Svärd et al. [112] defined five fundamental attributes of migration, ranging from performance and resource consumption to the ease of application, and evaluated pre-copy, post-copy, and hybrid approaches based on these criteria. Nathan [89] et al. proposed a model for estimating the duration of live migration, considering the dirtied memory size, the number of pages whose transfer can be deferred,

and the relation between these two factors over the course of migration. Breitgand et al. [31] analyzed the relation between the pre-copy duration and the service quality during migration, and formulated a cost function to be used for the optimal combination of pre-copy duration and down time. Akoush et al. [23] presented simulation models for estimating the total duration and down time, particularly on the Xen platform. Their models take parameters such as the network bandwidth and page dirty rate into consideration, and provide predictions of the migration duration within 90% accuracy.

Memory-Intensive Workloads

Live migration under memory-intensive workloads has gained attention as another challenging problem in clouds. Ibrahim et al. [63] identified problems with heuristics used in qemu-kvm's live migration when used with scientific computing workloads, such as MPI and OpenMP. They proposed a dynamically adjusted algorithm reflecting the rate of memory content mutation, thereby improving the down time and reducing performance degradation of the VM. Shribman and Hudzia [108] investigated a number of techniques applicable to pre-copy and post-copy migration for memory-intensive workloads. These techniques included the reordering of pages during transfer, delta encoding of page contents, use of RDMA, demand pre-paging, and explicit support for non-blocking page faults.

Application-Level Migration

There also exists previous work that takes the task of migration into the application layer. Zephyr [47] is an example of such approaches applied to online databases. It targets a shared nothing database architecture, and preserves ACID properties during migration. Madeus [86] is also a migration scheme for databases, which efficiently handles databases with heavy workloads by concurrent dissemination of their snapshots. ElasTraS [44] uses live database migration to build highly elastic and scalable database management systems. Imagen [79] targets active sessions for JavaScript web applications, migrating them between devices for ubiquitous access. Wang et al. [116] presented a proactive fault tolerance approach to preserving MPI applications, through triggering their live migration based on monitored performance.

2.2.3 Impact of Latency on Systems Usability

The impact of latency on system usability has been an active field of research since the early days of computer sciences. A survey by Rushinek and Rushinek [100] reported

that system response time is the most dominant factor that determines user satisfaction with systems. Studies on mainframe time-sharing systems also showed that increases in system response time lead to decreases in system performance [35, 43, 51], and the stress level of users [72]. More recently, studies on web pages and browser-based applications demonstrated a negative correlation between their response time and user satisfaction [59, 88].

Even before the above studies were conducted, in 1968, Miller introduced guidelines for response time requirements [84]. This work discussed a variety of response types and their time requirements over 17 different contexts. For example, responses to key typing need be delivered within 0.2 seconds, and turn-around time of script execution should be 15 seconds or shorter. Several studies in the 1970's and 1980's expanded these guidelines and aimed to make them more practical. While work by Guynes [53] and by Kuhmann [72] investigated text-based interaction, work by Goodman and Spence [48] involved graphical plots.

In the current era, research has progressed to suggesting thresholds over response delays that affect user experience [42, 66]. Ng et al., for example, found that users can perceive delays as low as 2.38 ms while performing dragging actions on a touchscreen [91]. While evaluation based on such thresholds is useful, we take a direct approach to evaluating usability in Chapter 4, by having human subjects interact with working systems.

In the systems community, evaluation of network latency on thin clients has been performed. Tolia et al. [113] measured the interactivity of thin clients by means of per-operation delays, testing applications ranging from image editing software to office productivity tools over a range of network round-trip time and bandwidth. Lai and Nieh [74] used slow-motion benchmarking, which quantifies the delays in interactive events through monitoring the network traffic corresponding to these events.

Virtualization facilitates the relocation of thin-client servers for providing better interactivity to end users. THINC [27, 68] virtualizes the display driver interface, and translates the native commands to simplified operations amenable to client-side hardware support. MobiDesk [28] efficiently supports virtual desktops in mobile environments by decoupling the user's workload from host systems and devices, such as the display, OS, and network, and facilitating its migration between hosts. VMShadow [52] controls the locations of VMs in distributed clouds to reduce network latency to their users, by deciding their placements based on monitored performance of their network traffic.

2.2.4 Execution Knowledge and Proactive State Transfer

The knowledge about VM status has been captured and used in different forms to achieve various objectives, including those other than VM state transfer. In addition, forms of proactive state transfer have been studied outside the area of virtualization.

Enlightenment

The key aspect of our work on enlightened post-copy is *enlightenment* [85], knowledge provided by the guest and exploited by the hypervisor for efficient VM management. Enlightenment has been used in various ways. Satori [85] uses the knowledge of guests about their reclaimable memory, for memory consolidation between multiple VMs. Ballooning [115] is another well-established form of explicit guest involvement in the memory reclamation by the hypervisor. Our work applied the concept of enlightenment to migration, and investigated how explicit guest support can improve migration performance. An alternative approach to full enlightenment through guest cooperation is to use hypervisor-level inference. Kaleidoscope [33] exploits memory semantics inferred from architecture specifications, and uses the obtained information for fast VM cloning. JAVMM [58] expedites migration of VMs containing Java applications, by having them inform the hypervisor of memory containing garbage-collectable objects and avoiding its transfer.

Proactive State Transfer in Other Areas

The concept of proactive state transfer has been studied extensively as data prefetching in the area of file systems. Access transition between files or disk blocks, with associated probability, has proven effective in making prefetch decisions [50, 76]. Speculative execution [39] predicts future disk accesses in an automated fashion by executing code ahead of time using spare CPU resources to find I/O requests. Automatic generation of information about disk accesses is also done at the compiler level [32, 87], where application code is annotated to generate prefetching directives. Finally, applications can explicitly disclose their file access patterns to achieve high effectiveness. Examples of this approach include informed prefetching [69, 93, 114] and application-controlled prefetching [36, 37].

Chapter 3

VM Delivery over Wide-Area Networks

The first of our problem contexts is the delivery of VMs, specifically in the form of virtual appliances, over WANs. Virtual appliances are pre-configured VM images with user applications already installed for use. While widely used by today's cloud users, they are as useful for end users located across WANs as a means of software delivery. Virtual appliances ensure the executability of the applications in the user's environment. Furthermore, software packaging through VM encapsulation removes the burden, on the user's side, of manually meeting requirements and maintaining the user's native environment, such as installing dependencies of the new application. However, the size of virtual appliances can often be discouragingly large for timely transfer over WANs; when the size is in the order of tens of gigabytes, downloading the entire VM image can take up to hours. Unfortunately, such time-consuming download is the common way of obtaining new virtual appliances as of today.

In order to provide easy access to virtual appliances for end users, an efficient streaming approach is essential. Similar to video streaming, such an approach streams the image of a virtual appliance as the user comes into the service; the user visits a repository of virtual appliances and selects one of interest. Then, the VM is launched rapidly on the user's machine, with continuous execution with adequate performance. The user can also switch to another virtual appliance and stream it, when finished with the current one. While this capability is attractive, VM streaming is more challenging than video streaming; VM execution is inherently non-deterministic, and the order of VM state accesses depends on the current workload. Therefore, VM state cannot be simply streamed in a sequential manner as done with video contents. Furthermore, the rate at which the state is accessed also vary widely. Specifically, the state access is typically bursty in nature. An application inside the virtual appliance, for example, may load a large amount of data from its disk into memory

at one point, and continue executing without accessing much of additional data for an extended period of time. For these reasons, an efficient streaming approach needs a way of predicting VM state accesses, and performing state transfer accordingly.

This chapter describes our approach to streaming virtual appliances that achieves timely launch and smooth execution of the VM through the use of execution knowledge. The core idea is using the knowledge about past instances of a given virtual appliance, and applying predicted state access patterns to the stream sequence. Key properties of virtual appliances make past execution knowledge particularly useful in achieving the timeliness and satisfying additional objectives, such as bounding the amount of state transfer. Our system, called `vTube`, demonstrates the effectiveness of using past execution knowledge in VM streaming, and its feasibility over networks with low bandwidth and high latency including 3G, 4G LTE, and public Wi-Fi's.

3.1 Problem and Objectives

The problem of VM delivery over WANs in our context poses primary challenges in timeliness. We elaborate on our use cases that lead to these challenges, define our objectives through metrics of importance, and explain how we apply execution knowledge in the domain of this problem.

3.1.1 Use Cases

Virtual appliances for end users provide various benefits through seamless delivery of new applications. Notable examples include application packaging, media content distribution, and application streaming to local data.

Application packaging: Virtual appliances allow software distributors to package an application together with its dependencies, ranging from even an OS to user libraries. By encapsulating them in a VM, virtual appliances can ensure the executability of the target application on the client side. This holistic packaging eliminates the burden of manual installation of these dependencies on the user, or the requirement of having a particular platform for running the application. For example, Windows applications encapsulated in a VM do not require that the user has a native Windows environment, as long as a compatible hypervisor is available.

Media content distribution: Virtual appliances provide encapsulation for not only com-

putation, but also data. Specifically, media contents requiring special handling can be placed in a VM. A video content in an exotic format, for example, can be coupled with its codec, or a movie with DRM protection can be shipped with an appropriate license. Such encapsulation becomes particularly useful as the bundled entity can often be of temporary use with the media content itself. When the user is finished with the content, the accompanying codec or license is no longer necessary. Disposing the VM removes both of them, without leaving unnecessary installations in the user's machine.

Application streaming to local data: Software in a virtual appliance enables computation to be moved to the site of a large volume of data, avoiding the difficulty of moving the large data to a computer with the appropriate software. The user may, for example, have a large volume of video files in a local computer. When the computer does not have a media editing application for the videos, one can be streamed in a virtual appliance. In this case, the data conveniently remains even after the software is removed.

Software Archiving

Archival of executable contents, such as Olive [105], in the form of VMs exemplifies the application packaging use case. Encompassing the benefits of VMs above, it transforms their preserved executability into a vehicle for easy access to historical software. Such archiving services play the role of libraries for digital contents, analogous to those for books, that maintain the contents for the coming ages. For example, educational programs for learning geography on DOS can be executed on today's computers precisely as they were in their time. The strength of VM encapsulation is that the target software remains executable despite the loss of the compatible platform in the latest environments, which makes virtualization a core requirement for the archiving services. In Olive's model, central repositories provide a collection of archival VMs, as depicted in Figure 3.1. In order for these repositories to deliver to end users across the Internet, an efficient mechanism for streaming VMs over limited bandwidth and long latency is critical.

Design Assumptions

The driving use cases above lead to a number of design assumptions. First, downloading entire virtual appliance images is not affordable when the capability to rapidly instantiate and interact with a VM is a key requirement. Therefore, we cannot relay on the straightforward approach, which is commonly used by existing services today, such as

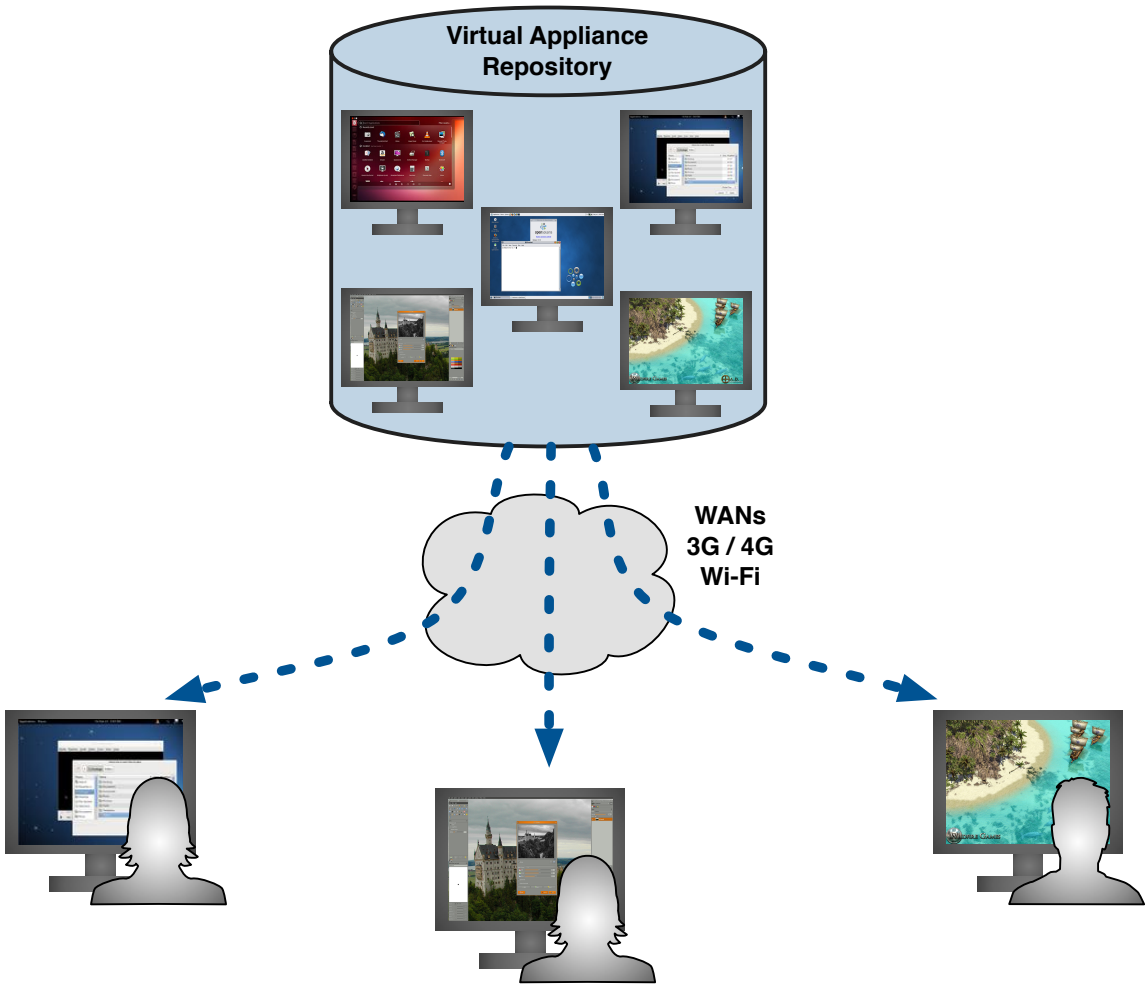


Figure 3.1: Repository of Archival VMs

VMware's Marketplace [107]. Second, we do not assume special construction of virtual appliances, other than that they are made for the use of specific software. Specifically, we do not enforce any guest changes or require effort to make virtual appliance size as small as possible. Third, the virtual appliance images include a snapshot of an executing VM, with a memory image. While omitting the memory state may reduce the virtual appliance size, it would induce boot time of the VM on the client and increase the initial wait time for the user. This extra wait time would degrade the user's swift interaction with the streaming service. Finally, the VM delivery model is based on a thick-client model, in which the capability to compute is retrieved from a remote site and executed locally. A contrasting approach is thin clients; in thin clients, the computation is performed on the remote site and only the user interaction handling is processed to the client side. We cover the topic of evaluating our thick-client model against the thin-client model in Chapter 4.

3.1.2 Goals: Metrics of Importance

VM delivery over WANs caters to end users launching virtual appliances to access software that their environments do not originally have installed. To this end, a practical system needs to address two primary metrics of performance: VM launch time and execution quality.

VM launch time:	The time it takes for the VM to start on the client side. It corresponds to the time for having VM state delivered that suffices for adequate execution performance.
VM execution quality:	The performance of the VM after it initially launches. It includes any disruption of its execution resulting from ongoing state delivery.

These metrics together represent the accessibility of virtual appliances over low-bandwidth, high-latency networks. A successful approach copes with the sheer size of VM images in the order of tens of gigabytes, and makes it feasible to instantiate them rapidly, within minutes, even with limited network resources for state transfer.

Additional metrics underneath these primary metrics include state transfer amount and VM stalls due to pending state arrival. Given particular bandwidth, the amount of state transfer decides the wait time for the user. The more effective the state transfer is in delivering what is required for the VM's smooth execution, the less time it takes until the VM is ready. Furthermore, the efficiency can be crucial for the delivery over certain types of last-mile networks, such as 3G and 4G cellular networks, that are charged per data usage;

downloading tens of gigabytes is usually very expensive on such networks. VM stalls are the disruption of VM execution that occurs when the guest is accessing state that has not been cached locally. Until the corresponding state has arrived, the VM execution thread must be suspended. These stalls, as the VM has already launched, directly transform to the disruption of the user interaction. Therefore, reducing state transfer amount and VM stalls is a key to improving VM launch time and execution quality.

3.1.3 Past Execution Knowledge

Our approach to addressing these challenges is the use of *past execution knowledge*, namely the information about the previous execution instances of a given virtual appliance. Specifically, two key properties of virtual appliances motivate us to apply this form of execution knowledge:

- They are static VM images, whose starting point of execution upon delivery is fixed.
- They are typically constructed for well-defined workloads of particular software.

Virtual appliances are distributed as VM images containing the state of a VM at the time of their creation. Therefore, VM instances created from a particular image always start execution from the particular state captured by the image. This includes the state of its disk, memory, and all the other devices such as virtual CPUs and peripheral devices. Consequently, the divergence in VM state access across instances of the same virtual appliance results from the workload after their start, but not from differences in their initial state. In addition, the purpose of virtual appliances is facilitated deployment or use of software installed on them. For example, a virtual appliance for professional media editing software would most likely be used for the particular application, and not for installing unrelated software such as a web server application. This makes it tractable to infer the workload of a current instance from that of previous instances. These two aspects makes past execution knowledge particularly suitable for identifying an efficient way of streaming VM images.

3.2 vTube: Efficient Streaming of Virtual Appliances

We developed a system, called `vTube`, as a solution to efficient streaming of virtual appliances over WANs. `vTube` adopts the video streaming paradigm and applies it to streaming virtual appliances. Specifically, it uses three modes of VM state transfer: buffering,

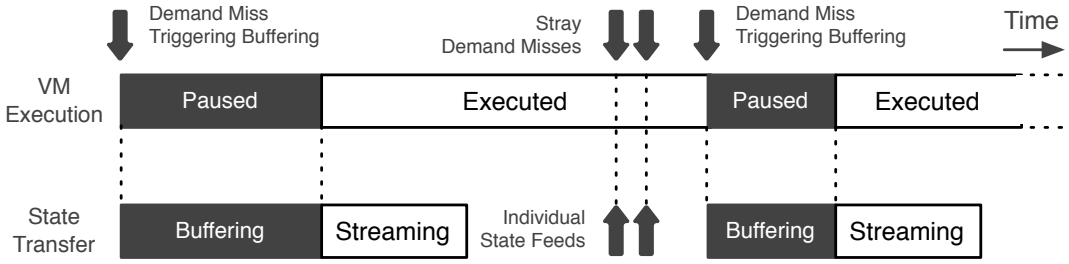


Figure 3.2: vTube Streaming Model. The top row represents a timeline of VM execution, and the bottom row that of VM state transfer.

streaming, and demand fetch. Figure 3.2 illustrates these transfer modes. Buffering is performed before VM launch or while the VM execution pauses. The rationale is to suspend the VM when the state transfer cannot catch up with its execution speed, analogous to the way video buffering is done. Otherwise, the user would suffer intermittent VM execution, and hence degraded user experience. `vTube` switches to streaming when it is affordable to transfer state in the background of VM execution. Streamed state is expected to arrive at the client side before it is accessed, without disturbing the VM when the corresponding access occurs. `vTube` allows state demand fetch when buffering or streaming has not delivered the state.

3.2.1 System Workflow

The design of `vTube` is based on a cycle of accumulation of past execution knowledge, generation of streaming guide information, and user sessions, as shown in Figure 3.3. Given a set of execution traces for a particular virtual appliance, the system performs processing called *clustering analysis*. This processing updates the knowledge about access patterns for the virtual appliance. Then, in each user session, the latest access pattern knowledge is dynamically applied to the decisions on streaming the VM state. The client records VM state accesses during each session, and uploads a new trace to the system at the end of the session. Clustering analysis is performed in an offline fashion as new execution traces come into the system. The latest version that is available is used for the current user session. Initially, the system can start with a small collection of traces for a given virtual appliance, and it accumulates execution knowledge as it handles user sessions.

Clustering analysis converts raw traces into a form of expression that can be efficiently handled at the time of streaming. Each VM execution trace consists of accesses to the

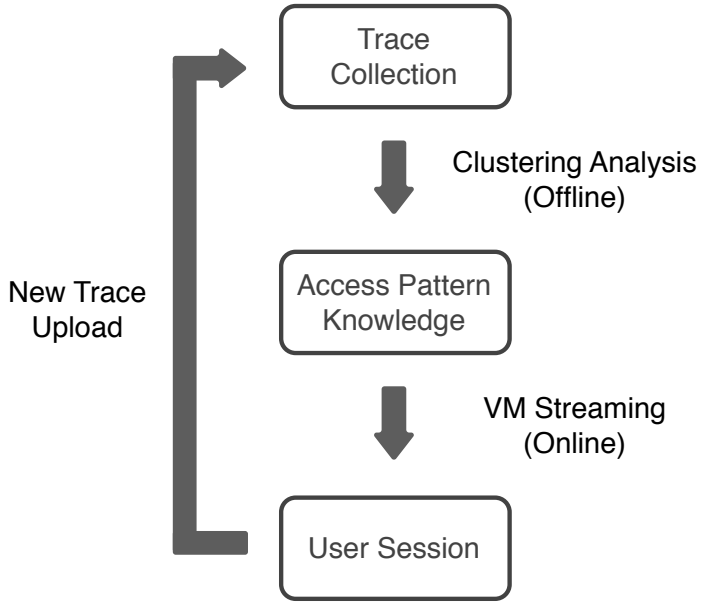


Figure 3.3: System Workflow of vTube

memory and disk, time-stamped relative to the start of the session. Following the common access unit of page size on x86 architectures, each memory and disk access is recorded at the granularity of 4 KB. We call this unit *chunk*. The chunk accesses are then grouped into variable-size units called *clusters*. Clusters are the minimal unit of state transfer in `vTube`, and they collectively form a stream sent to the client. For example, a cluster can consist of hundreds of memory chunks and tens of disk chunks; at the time of state transfer, these chunks in the cluster are always delivered together, instead of being considered individually for streaming decisions.

Behind the model of trace analysis and dynamic streaming exist our insights into VM state access in execution instances. Figure 3.4 shows a comparison between two execution traces of a virtual appliance containing Riven [16], an adventure game on Windows. The red and blue solid lines represent the cumulative amount of VM state accessed over time, for the two different traces. The dotted lines indicate the part of the accessed VM state from the solid line with the matching color that also appeared in the other trace, namely the commonality between the two traces accessed over time. As the figure shows, there exists a significant portion of VM state in each trace that is shared with the other. This overlap exists despite the fact that there was enough variability in the game play in the two traces. Also, each of the traces has a non-trivial amount of VM state access that is not

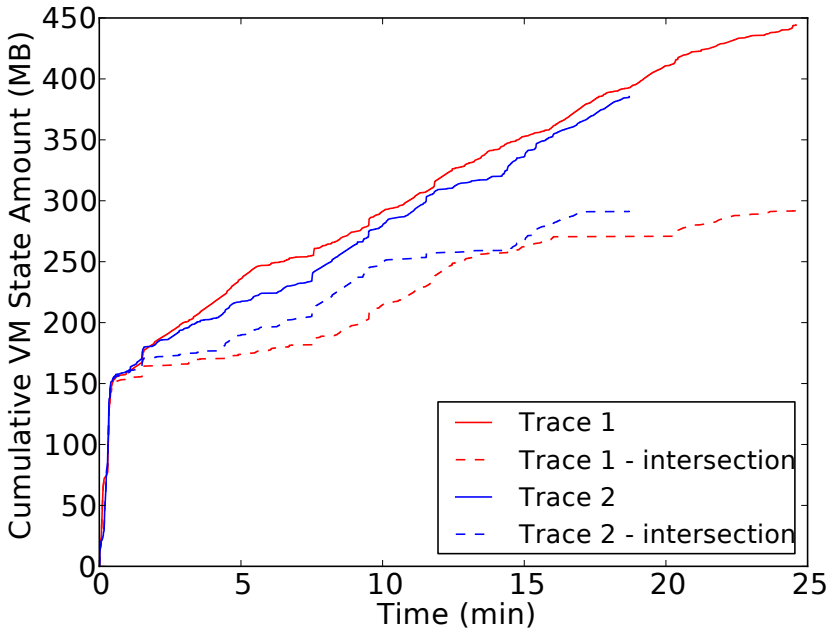


Figure 3.4: Example of VM State Access Commonality between Two Traces

seen in the other, which increases over time. This divergence between traces means that an efficient method needs to consider the current execution and adjust streaming decisions accordingly.

Figure 3.5 illustrates how clusters are accessed over time in two traces of the Riven virtual appliance. The top and bottom graphs each show a distinct trace, and the x-axis indicates cluster IDs having unique numbers across the traces. The circles represent clusters, with their size corresponding to the cluster size. The blue clusters are common to both of the traces, while the red ones appear only in either trace. The graphs show that the common clusters are accessed in varying orders and at different times in the traces. Furthermore, there also exist some short patterns that are common to the two traces. For example, a large cluster and several small clusters following it appear in the beginning of both traces, although their access timings slightly vary.

The nature of VM state accesses observed in Figures 3.4 and 3.5 leads to important implications for the design of an efficient streaming algorithm. First, the commonality across execution instances of the same virtual appliance validates the effectiveness of using past execution knowledge to infer VM state accesses of the current execution. Second, the

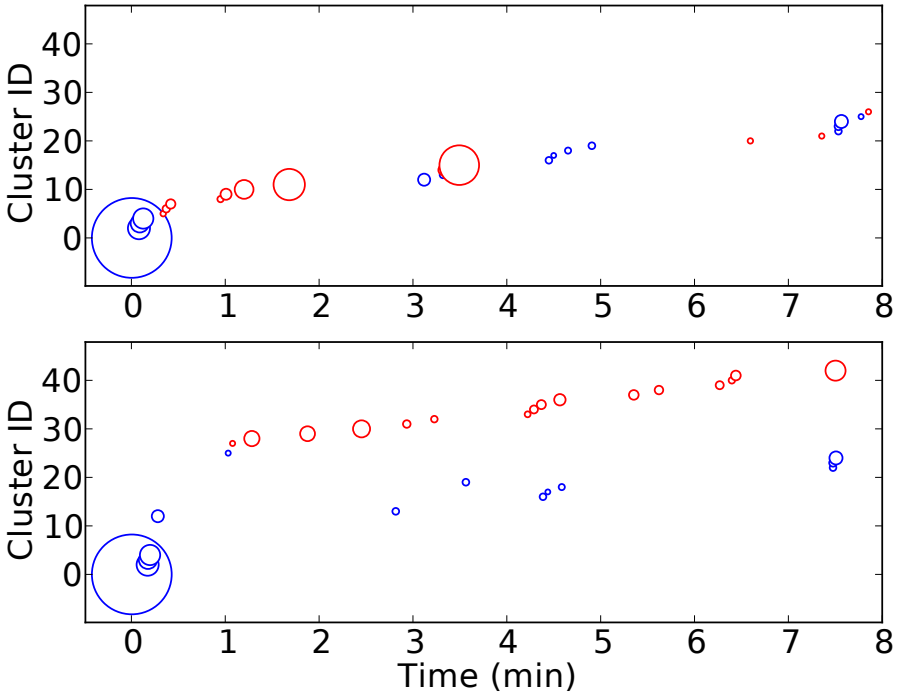


Figure 3.5: Example of VM state Access Orders in Two Traces

varying access timings and order of clusters motivate the use of streaming decisions that are adequately fine-grained. Otherwise, missed or out-of-order delivery of VM state would cause significant VM stalls. These observations drive the design of `vTube`'s streaming algorithm described next.

3.2.2 Streaming Algorithm

The rationale behind clusters is for them to be the unit of VM state transfer that achieves efficiency by capturing the nature of VM state accesses at the appropriate granularity. As the minimal unit of VM state transfer, each cluster should contain chunks that are essentially accessed together in time. This grouping brings two important benefits. First, clusters encompass the burstiness of VM state accesses, and allow treating them as a more coarse-grained entity. This amortizes the cost of dynamic streaming decisions over a number of chunks, compared to streaming individual chunks. Second, clustering also reduces the

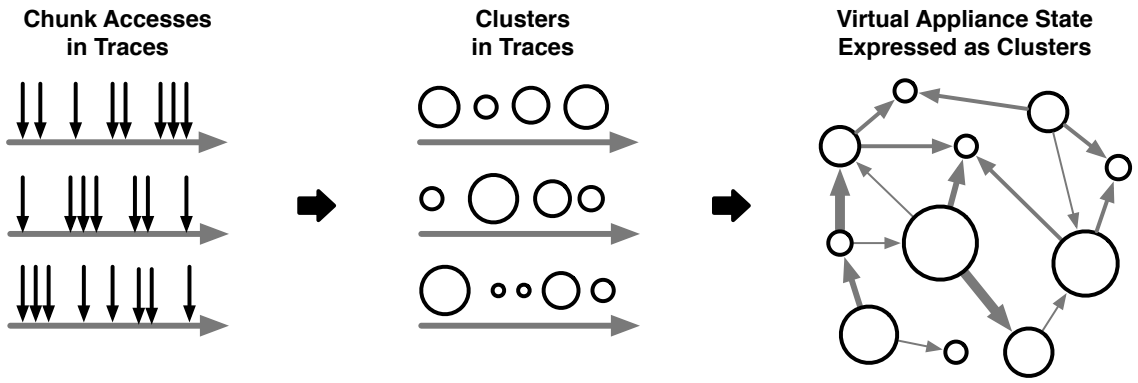


Figure 3.6: Overview of Clustering Analysis

complexity of VM access pattern analysis. As each trance can contain tens of thousands of chunk accesses or more, analyzing their access relations is otherwise hardly tractable on commodity hardware. Clustering thus prevents the handling of VM state accesses in `vTube` from being too fine-grained. On the other hand, clusters that are too large sacrifice streaming efficiency as they are more likely to include chunks that may not be accessed. Therefore, clusters have only tightly related chunks and serve as the unit that is neither too fine-grained or too coarse-grained. A majority of clusters typically contain one thousand chunks or more, having a size of 4 MB or larger, while some clusters can have a size in the order of kilobytes or tens of kilobytes.

Reflecting the nature of VM state accesses, the overall streaming algorithm of `vTube` is designed as follows: 1) it converts individual accesses in the traces to per-trace clusters, 2) these clusters are compared across the traces and formed into final clusters, 3) the algorithm derives access relationship between each pair of the final clusters, and 4) it delivers clusters to the client based on the current cluster access and its access relation to other clusters. Steps 1) through 3) belong to clustering analysis, summarized in Figure 3.6, and step 4) comprises the dynamic streaming during each session.

Clustering Analysis

In the first step of clustering analysis, we group the chunk accesses to per-trace clusters as shown in Figure 3.7. We form each per-trace cluster as a collection of chunk accesses that appear in the trace one after another within *clustering interval*. This interval excludes the time spent for the transmission of chunks, and we empirically set it to 2 seconds in

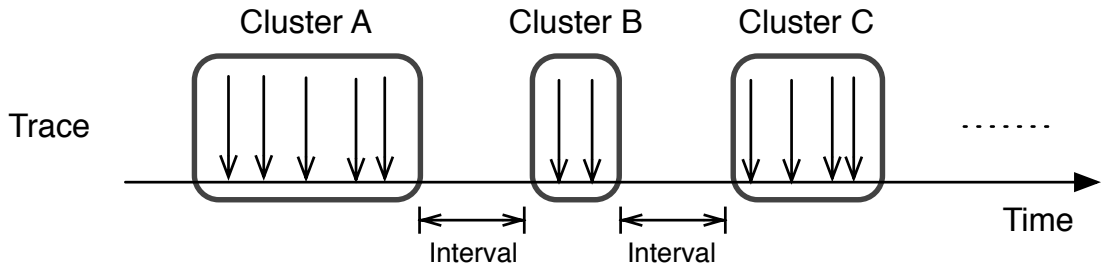


Figure 3.7: Clustering in Individual Traces

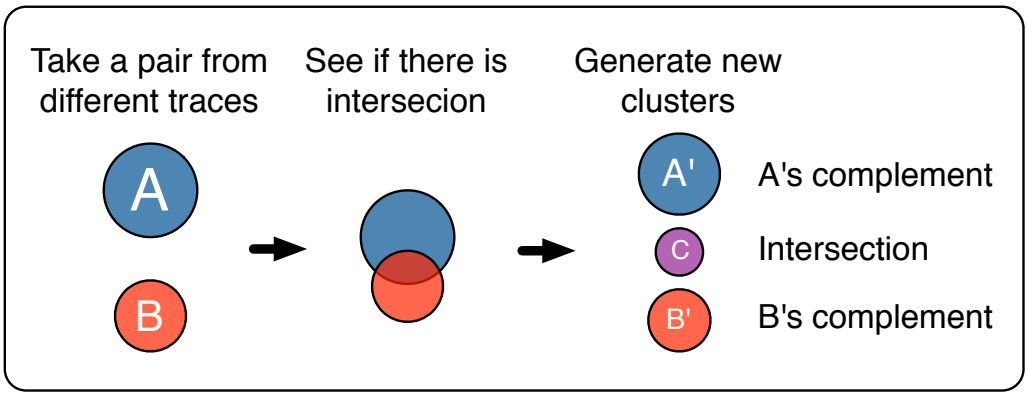


Figure 3.8: Clustering across Different Traces

our implementation of the system. The purpose of per-trace clustering is to group memory and disk accesses that are likely to be accessed together, as candidates for final clusters.

Next, we process per-trace clusters across traces. As per-trace clusters are specific to their trace, we compare them and generate clusters that are unique across traces. The process of forming new clusters out of two traces is shown in Figure 3.8. For each pair of overlapping clusters, we emit their intersection and the compliments of the original clusters as three new clusters. We repeat this process over the entire collection of traces, and generate final clusters that represent the maximum groups of chunks that are always accessed together.

Once the final clusters have been derived, the last step defines the relation between each pair that reflects the access patterns observed in the traces. This relation has two properties: *access probability* and *access interval*. Access probability is the likelihood of one cluster following the other, and is calculated from the number of times the condition

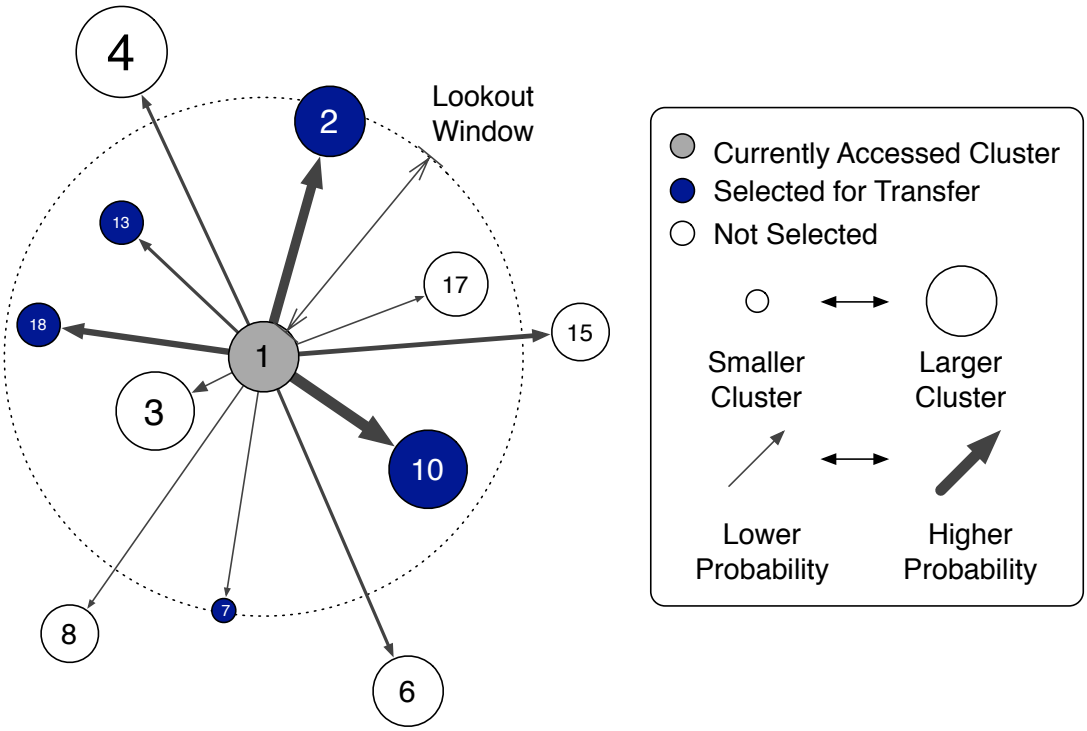


Figure 3.9: Cluster Selection for Delivery

holds in all the traces. Access interval is the expected time of access to the cluster since that of the other cluster. We select the shortest interval observed in the traces as this interval, to be conservative about state delivery and minimize the chance of late delivery that would otherwise occur if, for example, we used the average interval. With the relation defined between the cluster pairs, the state for the target virtual appliance is expressed as an inter-related collection of clusters. Dynamic streaming extracts expected VM state access patterns from this representation.

Dynamic Streaming

`vTube` makes dynamic decisions on what clusters to deliver to the client based on the current access by the executing VM. When a particular cluster is accessed, the algorithm first determines a set of clusters to be transferred, and then controls the VM execution with respect to the timings of their transfer. Figure 3.9 illustrates the selection method of clusters to be transferred along with the currently accessed cluster. In the figure, the gray

circle in the center represents the cluster currently being accessed, and the other circles the clusters that have relation to it. The circle size reflects that of the cluster itself. The arrows from the current cluster to the other clusters indicate the relation between them, with their length and width reflecting the interval and probability, respectively. In order to avoid looking infinitely into future cluster accesses, we first apply a threshold over the intervals, called *lookout window*. Then, among the clusters that do not exist on the client side and that have interval within lookout window, we select those that have probability greater than a shifting threshold defined as follows:

$$P(Y|X) > Percentile(Size(Y)) \tag{3.1}$$

This definition means that cluster Y is selected for transfer with current cluster X if its probability exceeds the percentile of Y's size based on the sizes of all the clusters sorted in an ascending order. Conceptually, the shifting threshold penalizes larger clusters more by requiring them to have a higher probability for selection. Large clusters typically correspond to more workload-specific and deterministic VM state access than small clusters, which appear in traces in a more random, unpredictable manner. In addition, transferring a large cluster costs more than a small cluster, in terms of time and bandwidth consumption. Therefore, the algorithm aims to be conservative towards deciding to send large clusters, while it can be optimistic and aggressive about small clusters. Figure 3.10 shows an example of cluster size distribution for a virtual appliance containing Riven. The cumulative probability of the cluster size 4 MB, for example, is approximately 0.5; if the size of cluster Y is 4 MB, its probability thus must be greater than that for Y to be selected for transfer with X.

Once the set of clusters is derived, we decide how to deliver them to the client with respect to VM execution. As depicted in Figure 3.11, the objective is to buffer just enough clusters that let the VM continue execution smoothly. In order to do so, we first order the clusters at their intervals from the current cluster and compute the cumulative bandwidth that is necessary to deliver them on time; the bandwidth requirement at a particular time is calculated as the aggregate size of all the clusters that need to arrive by that point, divided by the time we have. Depending on the cluster sizes, this estimate may not be monotonically decreasing as we look further into the future. We determine the last point at which the required bandwidth exceeds the bandwidth currently available to the client, and decide to buffer all the clusters appearing up to that point. Once these clusters have been delivered to the client, the required bandwidth for the remaining clusters stays below the current bandwidth. Thus, we resume VM execution and overlap it with the transfer of these clusters, as we can expect them to arrive by their access time.

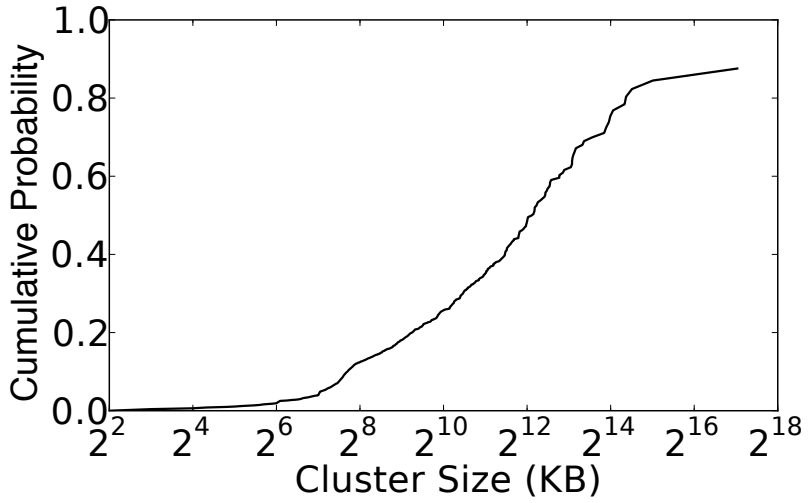


Figure 3.10: Example of Cluster Size Distribution for Riven Virtual Appliance

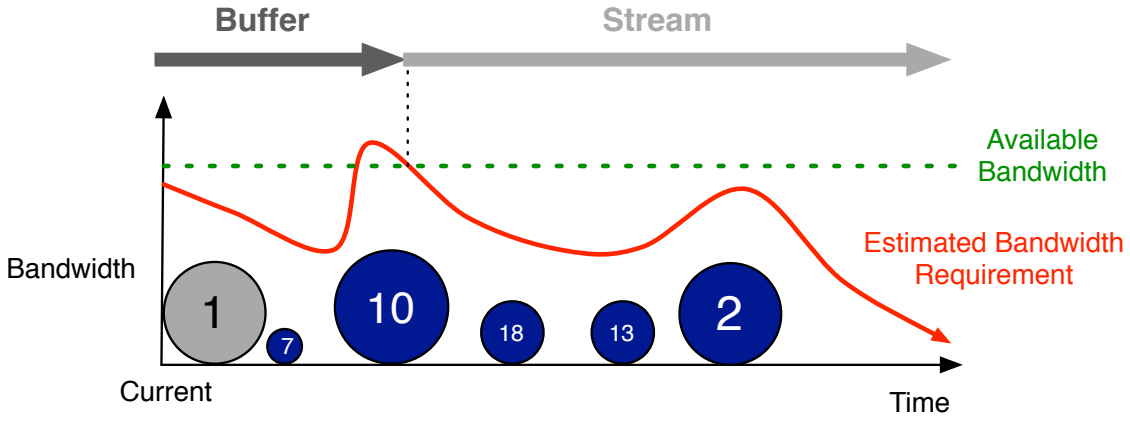


Figure 3.11: Cluster Transfer with Execution Control

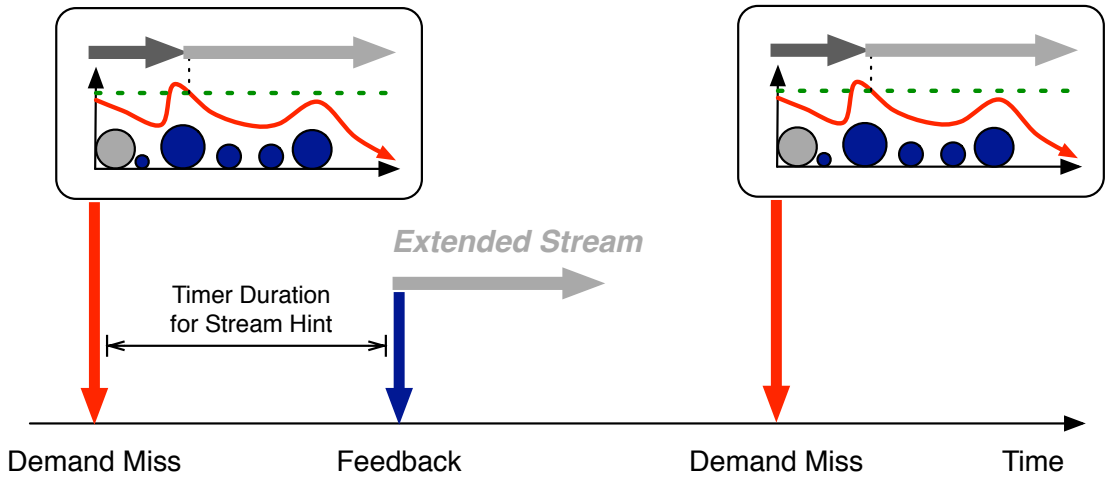


Figure 3.12: Cluster Streaming in User Session

Figure 3.12 shows the overall view of VM streaming during the course of a user session. The streaming decisions explained above are triggered by each demand fetch of a cluster. We also use an additional mechanism that extends the streaming, called *streaming hints*. Streaming hints address the fact the trigger by a demand fetch is reactive and streaming is performed only when the client accesses a missing cluster. However, streaming should continue when the estimation has been successful and, as a result, there is no demand fetch that triggers it. A streaming hint serves as an alternative to demand fetch in such a case, and notifies the server of the currently accessed cluster. They trigger the streaming decision process just as a demand fetch does, except that the current cluster does not need to be transferred and buffering is not initiated.

Demand Fetch of Individual Chunks

Each access to a chunk that does not belong to any cluster results in the direct retrieval of its content, without triggering buffering or streaming. This case occurs when the current VM execution accesses a chunk outside the coverage by the analyzed traces. The demand-fetch requests of such individual chunks are handled without explicitly controlling VM execution. Instead, the VM continues execution with ephemeral stalls while the requests are being served. Specifically, the demand fetch of a memory chunk can stall the execution of one of the VM's CPUs, while that of a disk chunk may be dealt with by the guest OS

without disrupting the execution. As we will show in Section 3.4.5, the demand fetch of individual chunks happens only infrequently.

3.3 Architecture and Implementation

vTube's architecture consists of 1) a server that, for each virtual appliance, maintains its VM image, access pattern knowledge, and a list of free memory chunks, and 2) clients in the form of a custom hypervisor that supports VM streaming. Figure 3.13 shows this architecture of vTube. The detail of the server and client implementation is described below.

3.3.1 vTube Server

The server has three kinds of data for each virtual appliance, which are generated by scripts that process VM images or traces. First, it stores the VM image including device, memory, and disk state in a format compressed on a chunk basis. Each chunk content is written with its SHA-1 hash. When sending chunks to the client, the server uses the hash to identify those whose content already exists on the client. For such chunks, the server informs the client of the hash values instead of sending their contents, which can be looked up in the client's content-addressable cache explained later.

Second, the server maintains access pattern knowledge as versioned files. Each file contains the results of clustering analysis, with member chunks, probability, and interval of each cluster. A new version is generated as a result of executing clustering analysis with the current set of traces for the virtual appliance. The server uses the latest version available at the start of a session.

Third, the server imports a list of memory chunks that are unallocated by the guest OS, when the virtual appliance is added to the system. The contents of these chunks are not necessary for the correct execution of the guest, and the server eliminates their transfer by sending this list to the client at the start of each session. When a chunk in this free memory list is accessed, the client returns a zeroed-out chunk without contacting the server. The list is obtained either with the help of a kernel module for Linux guests, or by parsing memory images to find zeroed-out pages for Windows guests.

The server's streaming functionality is implemented as a multi-threaded TCP server program. As the client generates chunk requests, the server makes a prediction of future state accesses using the dynamic streaming algorithm described in the previous section.

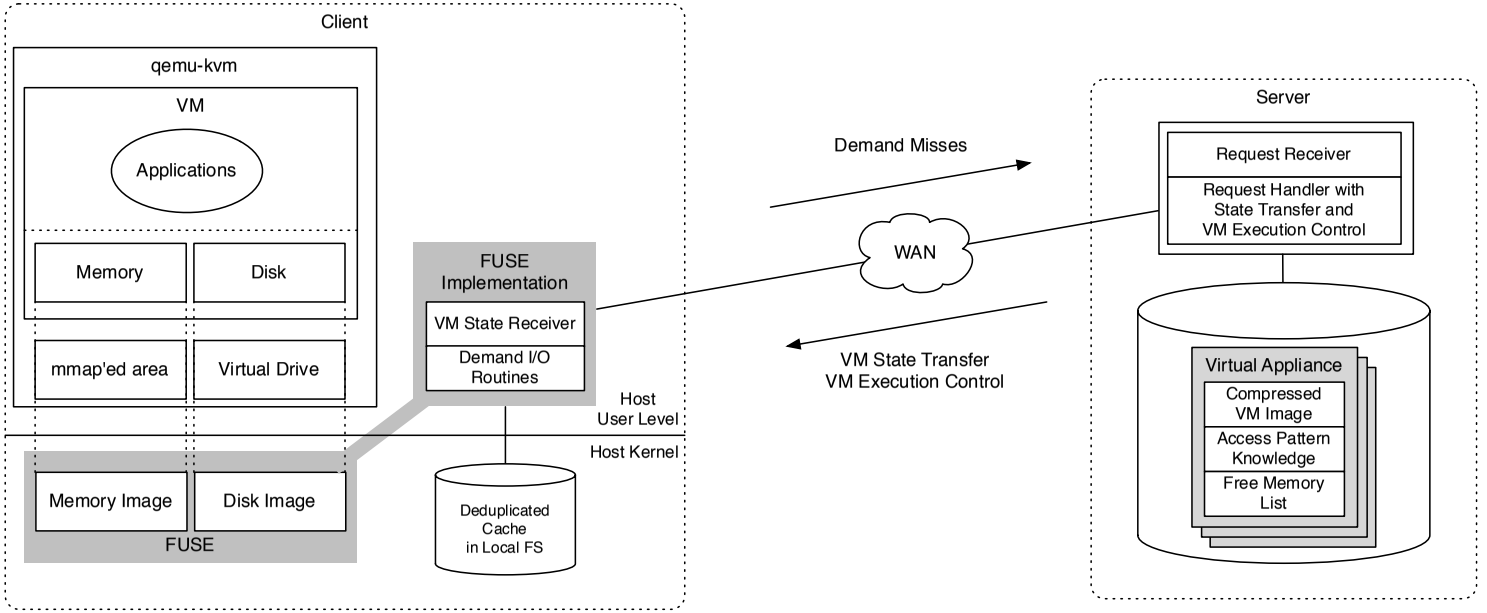


Figure 3.13: vTube Architecture

The server sends clusters according to the prediction, while also monitoring the transfer rate to the client. It decides whether buffering is needed using this rate as the currently available bandwidth, and if so sends a VM suspend message to the client. At the beginning of a session, the client's first cluster access sets buffering on. The currently available bandwidth is updated periodically, and when the server decides that it can switch to streaming it sends a VM resume message. For the rest of the session, the server continues transferring clusters on a demand fetch or streaming hint, making buffering decisions based on the latest measurement of the available bandwidth. State transfer and VM execution are thus controlled solely by the server; the client only communicates the current chunk accesses to the server, and does not incur other overhead.

3.3.2 vTube Client

The client is implemented as a modified version of qemu-kvm 1.1.1. It communicates with the server to initiate each session, issues requests for chunk contents to the server, and controls VM execution as instructed by the server. The client treats the VM state accesses at the granularity of chunks. The notion of clusters is transparent to the client. When the VM accesses a particular chunk, the client requests its content from the server. The server translates it to an access to the cluster that contains the chunk, and returns all the chunks of the cluster.

To the rest of the hypervisor, the client passes the VM's memory and disk state as regular files that are exposed through a custom FUSE implementation [5]. FUSE is a mechanism for implementing file systems at the user level, and it allows using the streamed state for the VM. Each chunk access is first directed to a content-addressable cache in the client's local file system. As the client receives chunk contents from the server, it stores them in this cache. When the content of the accessed chunk is available in the cache, the FUSE implementation returns it as the corresponding file content. Otherwise, the access results in a request to the server and it completes when the corresponding content has been received. The cache serves chunk contents across sessions and different virtual appliances on the same client. When starting a session, the client notifies the server of its current cache state; through this message, the server knows what contents already exist on the client and avoids sending duplicated contents over the network.

The client also records memory and disk chunk accesses by using the FUSE implementation during the session. The first access to each chunk is tracked in the I/O routines for the exposed VM image files. At the end of the session, the client makes a compressed file of the trace, and sends it to the server. This new trace is added to the server's collection for the virtual appliance, and is used for generating a new version of the access pattern file.

3.3.3 Session Handling

Each session in `vTube` proceeds as follows:

Step 1: When the user starts a new session, the client fetches the metadata of the virtual appliance, including its virtual hardware configuration and free memory list. The client also sends a list containing the hashes of the chunk contents in its content-addressable cache from previous sessions. The server selects the latest version of access pattern knowledge for the virtual appliance that is available at this time.

Step 2: The client starts the VM and generates the first chunk access, which triggers initial buffering. The server executes the dynamic streaming algorithm and decides what clusters should be buffered. After these clusters have been sent, a VM resume message follows and the VM resumes execution. The server avoids the transfer of any chunks that already exist in the client cache, and also keeps its own copy of the cache state to eliminate any duplicate contents sent over the network. Assuming that the client cache is initially empty and it has enough space, only the first access to a particular content results in its retrieval.

Step 3: While the VM executes, the client makes a demand fetch request to the server if the accessed chunk is not found in the cache and it is not in the free memory list. The server executes the dynamic streaming algorithm on the demand fetch request, and sends back the appropriate clusters. When buffering is necessary, it sends a VM suspend message before these clusters, and a VM resume request when switching to streaming. The server handles each chunk in these clusters in the following manner. If the chunk is in the free memory list, the server simply does not send it as the client can locally serve the corresponding access without its content. If the chunk content is stored in the client cache, the server sends its hash and the client returns the cached content that matches it. Otherwise, the server sends the compressed chunk content retrieved from the stored VM image. When the request contains a chunk not found in the access pattern knowledge, however, the server directly handles the individual demand fetch of the chunk.

Step 4: When the session finishes, the client uploads the trace to the server. Before sending the trace, it adjusts the timestamps by removing the wait time for network transfer. This processing creates a trace with idealized VM state access timings; the server takes this form of trace to generate access pattern knowledge, from which it can make dynamic streaming decisions based on the ideal VM execution performance.

3.4 Evaluation

We conducted experiments to demonstrate `vTube`'s timely launch of virtual appliances in environments with limited network resources. We first describe the VM execution behavior under `vTube` streaming and application-level performance of various virtual appliances we constructed. We then investigate the effectiveness of `vTube`'s VM state access prediction, and compare its performance to an alternative method used by VM distribution networks.

3.4.1 Set-Up

We used a range of network configurations, as listed in Figure 3.14. For measurements that require repeatability and stability, we used an emulated environment whose bandwidth and latency was controlled with Linktropy [1]. On this emulated network, bandwidth was set to 7.2 Mbps, which is based on the nation-wide average reported by Akamai [18, 75], and 14.4 Mbps. Round-trip time (RTT) was set to 120 and 60 ms, which falls in the range expected in WAN settings. In the rest of our measurements, we streamed virtual appliances over real networks including domestic 3G and 4G networks, two public wireless networks overseas in Taiwan, and a public wireless network in a local coffee shop. In all these cases, the server was located in Pittsburgh, Pennsylvania.

The server machine was equipped with an Intel Core i7 at 3.40 GHZ and 32 GB memory. The client laptop had a dual-core Intel Core 2 Duo CPU at 2.40 GHz and 4 GB memory. All measurements were done with a cold cache, i.e., no initial contents in the content-addressable cache on the client.

Virtual appliances: We constructed seven virtual appliances that span a range of workloads and platforms. All virtual appliances had 1 GB memory and a disk between 10 and 20 GB. Those with Linux ran Ubuntu 12.04. These virtual appliances have varying use cases, and we accordingly use different evaluation metrics.

1. *Mplayer*: a video player on Linux [10]. It plays an AVI vide file that is 5 minutes long and 84 MB in size.

2. *Avidemux*: a video editing application on Linux [3]. It converts eight MP4 files to AVI format. The files are locally supplied and 698 MB in total.

3. *Arcanum*: a 2D role-playing game on Windows XP [2].

Label	Type	Bandwidth	RTT
7.2 Mbps	Emulated	7.2 Mbps	120 ms
14.4 Mbps	Emulated	14.4 Mbps	60 ms
3G	Real	8-12 Mbps	52-113 ms
4G	Real	1-29 Mbps	96 ms
Taiwan (Good)	Real	7-25 Mbps	220 ms
Taiwan (Fair)	Real	4-9 Mbps	220 ms
Coffee Shop	Real	5-7 Mbps	14-175 ms

Figure 3.14: Summary of Network Configurations. Bandwidth and RTT for real networks indicate representative values, or ranges if high variance was observed.

Virtual Appliance	Description	OS	Total Size (Compressed)	Download Time (7.2 / 14.4 Mbps)
Mplayer	Video Playback	Linux	4.6 GB	87 min. / 44 min.
Avidemux	Video Editing	Linux	5.1 GB	97 min. / 49 min.
Arcanum	Game	Windows XP	6.9 GB	131 min. / 66 min.
Riven	Game	Windows 7	8.0 GB	152 min. / 76 min.
HoMM4	Game	Windows 7	5.6 GB	106 min. / 53 min.
Selenium	HTML Rendering	Linux	4.4 GB	83 min. / 42 min.
Make	Compilation	Linux	4.6 GB	88 min. / 44 min.

Figure 3.15: Summary of Virtual Appliances

4. *Riven*: a 2D adventure game on Windows 7 [16].

5. *HoMM4*: a 2D strategy game, Heroes of Might and Magic IV, on Windows 7 [6].

6. *Selenium*: an automation test script for `Firefox` [17]. The script browses through Python documentation in HTML format.

7. *Make*: compilation of `Apache` 2.4.4 source code on Linux. The source tree is locally supplied.

Figure 3.15 summarizes the properties of these virtual appliances, with the total size of their images after compression and download time over 7.2 and 14.4 Mbps. All virtual

appliances are in the order of gigabytes in size, and the corresponding download times are at least 42 minutes over 14.4 Mbps and up to 152 minutes over 7.2 Mbps.

Access pattern knowledge: In order to generate the server's access pattern knowledge to be used in our measurements, we manually collected 10 traces for each of the virtual appliance except for Riven and Make. For these two virtual appliances, we used 16 and 6 traces, respectively. When recording these traces, we intentionally introduced variability in using the virtual appliance. With Mplayer, different functionalities were used such as skipping, pausing, speed-up, in addition to normal playback of the video. With Avidemux, the tasks included video and audio conversion to a number of different formats. With Arcanum and Riven, game plays started at different points including the start of the game and some save data files. With HoMM4, tutorials at different difficulty levels were tried. With Selenium, browsing behaviors involved looking at different contents, searching for key words, and following links. Make was an exception, in which the compilation was consistently repeated.

3.4.2 Streaming Behavior

Figure 3.16 illustrates the streaming behavior of `vTube` with the Riven virtual appliance over the real networks. The black portions of each bar represent VM execution, and the gray parts buffering events. Additionally, there are VM stalls due to demand fetches; these are approximately equal to RTT between the server and client, and they typically do not significantly impact the VM execution experienced by the user. In all the cases, `vTube` launches the VM within several minutes, after which occasional buffering occurs. These additional buffering periods are no longer than 20 seconds. The 4G case, for example, has one buffering event longer than 10 seconds, five events between 1 and 10 seconds, and five events below 1 second. The general trend is that buffering becomes shorter and less frequent over time. Once the initial and following buffering periods complete, the VM execution continues smoothly.

For comparison, the figure also shows the time it takes to perform partial downloading of the virtual appliance image as red bars. It corresponds to the downloading time of all the chunks in the trace set, after which VM execution with close to ideal performance can be expected. Compared to `vTube`'s initial buffering times, these times are significantly longer. For example, the downloading takes over 15 minutes in the Coffee Shop case. `vTube` expedites the initial launch time with the modest cost of occasional buffering events afterwards.

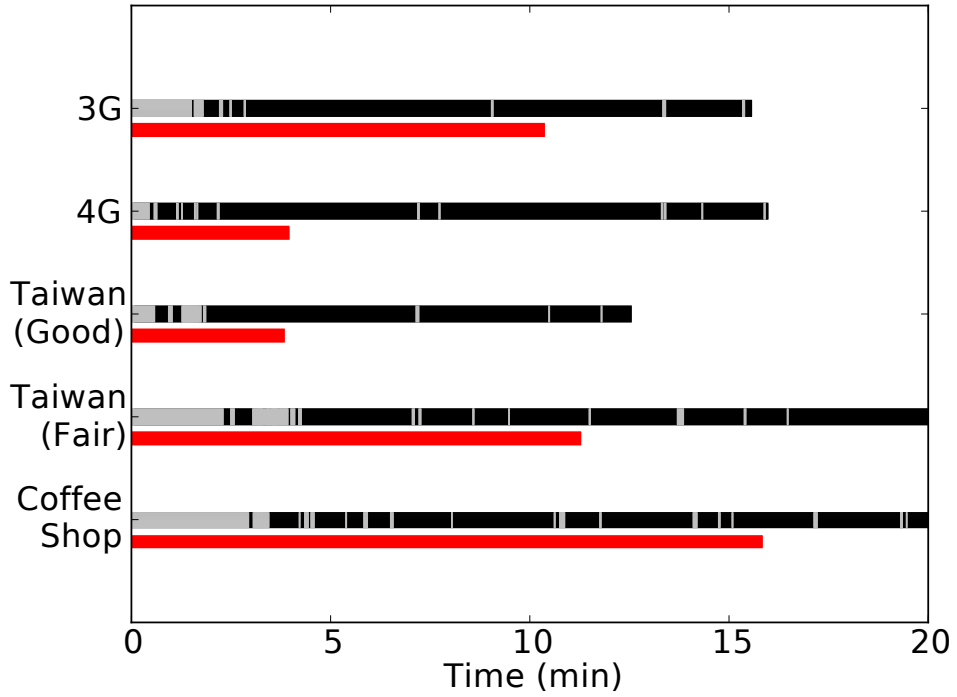


Figure 3.16: Behavior of Streaming Riven Virtual Appliance

3.4.3 Application-Level Performance

We measured application-level performance of four virtual appliances: frames per second for Mplayer, page traversal speed for Selenium, and completion time for Avidemux and Make. These measurements were conducted on our emulated networks. As baselines, we include results of local VM execution performance, and of demand-fetching over a direct wired link between the client and server. The wired case has very small RTT and thus little cost for chunk retrieval, and serves as a baseline that is close to the ideal VM streaming with perfectly accurate state transfer.

The results of Mplayer are shown in Figure 3.17. The y-axis indicates the video playback quality in the frames per second, and the x-axis elapsed time since the start of the session. We created the virtual appliance so that the video playback automatically starts after a few seconds of VM execution. The local case represents the upper-bound of playback performance, in which the frames per second stays at 25 right after the start of playback and throughout the duration of the video. In the wired case, Mplayer starts the playback af-

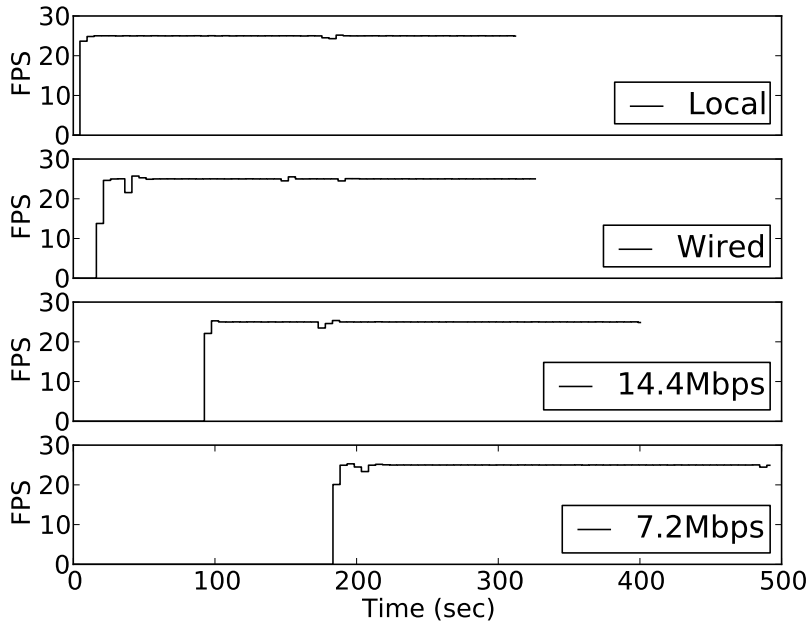


Figure 3.17: Mplayer Performance

ter a short delay of 16.3 seconds, during which the necessary VM state is demand fetched. The workload accesses a little over 300 MB of VM state, and `vTube` launches the virtual appliance with smooth execution in 92.5 and 183.3 seconds on the 14.4 and 7.2 Mbps networks, respectively. Once the playback starts, its quality is close to the ideal line with only occasional drops in the frames per second.

The Selenium results are shown in Figure 3.18. Similar to the Mplayer case, the Selenium launches automatically after the VM starts execution. In the figure, the y-axis shows the cumulative number of HTML pages traversed by the script, and the x-axis elapsed time. The rate of page traversal changes over time depending on the page contents and processing in the automation framework. However, the visited pages and their appearance order are fixed, and thus the shapes of the curves can be compared across the measurements. The curve in the local case represents the optimal rate of processing. The wired case, after a short delay, sustains the processing rate close to that of the local case. The workload generates a little over 150 MB of state access, and `vTube` launches the VM in 60.2 and 114.6 seconds on the 14.4 and 7.2 Mbps networks, respectively. Both of these streaming cases achieve the progress rate of the script similar to the ideal rate once the VM is launched.

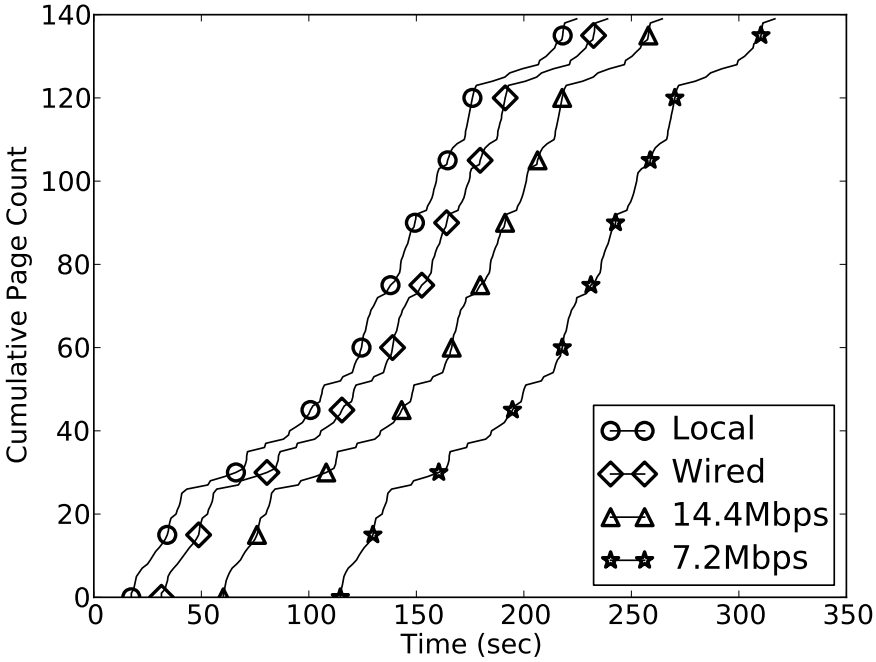


Figure 3.18: Selenium Performance

Table 3.1: Avidemux and Make Performance

Virtual Appliance	Network	Launch Time (sec)	Workload Completion Time (sec)	Overhead on Local
Avidemux	LAN (Demand-Fetch)	6	75	54.6%
	14.4 Mbps	26	67	39.5%
	7.2 Mbps	52	65	33.8%
Make	LAN (Demand-Fetch)	7	420	2.8%
	14.4 Mbps	17	407	-0.4%
	7.2 Mbps	32	411	0.4%

Table 3.1 shows the results for Avidemux and Make. It lists the VM launch time and completion time of these workloads, and the increase in completion time compared to the local case. With Avidemux, the wired and vTube over 14.4 and 7.2 Mbps cases launch the VM in 6, 26, and 52 seconds, respectively. While completion time is 75 seconds in the wired case with an increase of 54.6% compared to local execution, vTube reduces it to around 65 seconds with an increase of less than 40%. vTube achieves this reduction by streaming VM state effectively and making it mostly available locally for Avidemux. On the other hand, demand-fetching each chunk results in a higher increase in completion time. With Make, launch time is 7 seconds and the increase in completion time is 2.8% in the wired case. vTube spends 17 and 32 seconds at 14.4 and 7.2 Mbps, respectively, for initial buffering. As the workload is consistent with the traces used, it caches VM state effectively and and makes completion time as good as that of the local case. Note that the reduction by 0.4% in the 14.4 Mbps case is due to fluctuations of the workload's execution time rather than an improvement over local execution. In most of the Avidemux and Make results, vTube takes more time than the wired case since the start of the session and until the end of the workload, namely the sum of launch time and workload completion time. Besides the ideal setting of the wired case, which has high bandwidth and low RTT for chunk retrieval, the streaming accuracy of vTube also accounts for the discrepancy. As the dynamic streaming deals with the inherent uncertainty of figure VM state access, it transfers more than the exact amount of VM state accessed by the workload. In the next section, we further evaluate this accuracy of state access prediction and resulting overhead in the amount of state transfer.

The results of these four workloads, Mplayer, Selenium, Avidemux, and Make, demonstrate the rapid launch of the VM made feasible by vTube. Compared to the time of whole image downloading on the respective bandwidth shown in Figure 3.14, the launch time is reduced from tens of minutes or more to several minutes. This is a key performance aspect of vTube that achieves its goal of timely virtual appliance delivery to end users.

3.4.4 Efficiency of Dynamic Streaming

In the preceding sections, we demonstrated the overall behavior of vTube's dynamic streaming and its impact on application-level performance. We next show systems-level statistics of the VM execution with vTube. We investigate four metrics that capture different aspects of the VM execution and experience by the user: fetch ratio, buffering ratio, buffering rate, and miss rate. Table 3.2 shows the statistics of these metrics across the seven virtual appliances, streamed over our emulated network with 7.2 Mbps bandwidth and 120 ms RTT. The reported numbers in the table are the average of three measure-

Table 3.2: Systems-Level Statistics of vTube

Virtual Appliance	Workload Duration (sec)	Accessed State (MB)	Fetch Ratio (fetched / accessed state)	Buffering Ratio (buf time / total time)	Buffering Rate (# buf events / minute)	Miss Rate (# misses / # accesses)
Mplayer	500	316 (2.4%)	1.03	0.33	0.24	0.20%
Avidemux	127	103 (0.8%)	1.33	0.39	1.88	0.82%
Arcanum	920	87 (0.8%)	1.51	0.04	0.41	0.40%
Riven	1223	379 (1.8%)	1.49	0.21	0.46	0.70%
HoMM4	926	256 (1.2%)	1.41	0.15	0.37	1.96%
Selenium	332	155 (1.2%)	1.45	0.31	0.96	0.12%
Make	451	76 (0.6%)	1.01	0.06	0.35	0.66%

ment runs. The duration of the sessions ranges from approximately 2 minutes for the short batch-processing workload of Avidemux to over 20 minutes for the interactive workload of Riven. The amount of VM state accessed during these sessions is between 76 MB for Make and 379 MB for Riven. The numbers in parentheses in the Accessed State column show the fraction of the accessed state out of the entire VM image size; the small fractions corroborate the efficiency of `vTube`'s dynamic streaming compared to whole virtual appliance downloading. In the following, we describe each the four metrics.

Fetch ratio: Fetch ratio is defined as the amount of VM state delivered divided by the amount accessed by the guest. It shows the accuracy of state access prediction and the efficiency in network resource use. `vTube` aims to deal with the uncertainty of future VM state access without making fetch ratio high. In all the workloads, fetch ratio is kept 51% or lower. The ones with high variability in user activities, such as the three games and Selenium, have a relatively higher ratio than those with low variability. Make and Mplayer have particularly low fetch ratios of 1.01 and 1.03, respectively. Given the range of functionalities available in the applications and the non-deterministic nature inherent to the guest execution, `vTube` effectively predicts the state access pattern from the traces and limits the amount of state transfer over the network.

Buffering ratio: Buffering ratio is the time spent buffering with the VM suspended divided by the duration of the session. It indicates the fraction of time the user waits for

the VM to execute, during which he is prevented from interacting with it. The virtual appliances have buffering ratio between 0.04 and 0.39. Avidemux , Mplayer, and Selenium have the highest ratios because of the short duration of their workloads. The ratio is at most 0.21 in the other cases, meaning approximately 80% of time or more during the session is available for smooth user interaction. Arcanum, despite the variability in its workload, has the lowest ratio because of the fact that the state access footprint of the game is lower than the other applications per unit time.

Buffering rate: Buffering rate is the number of buffering events per minute. It shows how frequently the user interaction with the VM is interrupted. This metric complements buffering ratio and reveals the nature of the wait time: whether buffering is a few long periods or a series of a number of short periods, given the same total buffering time. With the exception of Avidemux and Selenium, buffering rate is less than 0.50; VM execution thus continues for a few minutes or more without major interruptions in these cases.

Miss rate: Miss rate is defined as the ratio of the number of chunks that are demand-fetched to that of all the accessed chunks. Miss rate represents how often the VM experiences ephemeral stalls outside buffering periods, caused by access to the chunks dynamic streaming has not delivered. The miss rate of the virtual appliances is 1.96% with HoMM4, which is the highest, and well below 1.00% with the other virtual appliances. Thus, the VM state access estimated based on the traces captures most of the state accessed during the session. Although the total amount of state retrieved by demand fetch in each case is not necessarily trivial, the individual misses do not cause significant disruption of the VM execution unless they occur in a rapid succession. These misses result from either misprediction by the algorithm, including that of access timings, or the trace coverage of the chunks accessed in the measured session.

3.4.5 Trace Coverage

`vTube`'s dynamic streaming algorithm is history-based; as the access pattern is derived from the set of available traces, the predictability of state access in the current session depends on their coverage of possible state accesses. For example, if the access to a particular chunk being accessed in the current session has not been observed in the past sessions, demand fetch is the only way of retrieving its content. For this reason, the relation between the number of traces and the corresponding miss rate is a key factor that decides the practicality of the approach. Figure 3.19 plots the miss rate of the seven virtual appliances when

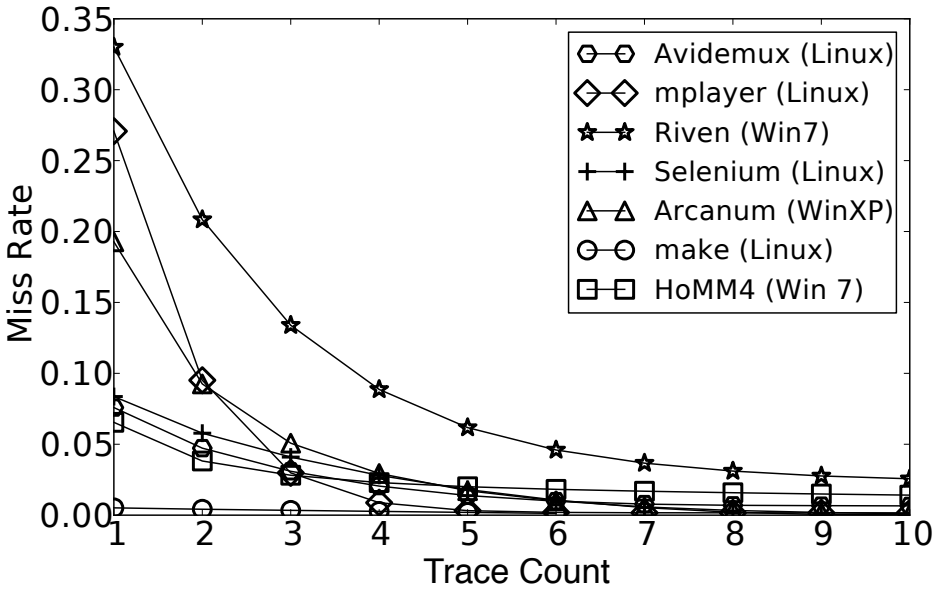


Figure 3.19: Miss Rates as Function of Trace Count

using varied number of traces to generate their access pattern knowledge. We computed the miss rates using the results of our experiments on emulated networks. For a given number of traces, the average miss rate of all the possible combinations out of the available traces is reported. As we add more traces, miss rate rapidly drops. With 10 traces, the trace coverage is high enough that miss rate becomes lower than 0.65% in all the cases except Riven and HoMM4, for which it is 2.6% and 1.4%, respectively. Make has a very low miss rate even with 1 trace, due to the low degree of variability in the workload. Note that we computed these numbers statically from the traces, whereas the measurements in the previous sections involved misses resulting from state arrival timings. Overall, the results in Figure 3.19 indicate that a relatively small number of traces suffices for providing good coverage of state accesses for all the virtual appliances. This effectiveness of traces in state access estimation makes it possible and practical to bootstrap the delivery of a newly added virtual appliance with a moderate number of pre-collected traces. Furthermore, `vTube` imports a newly generated trace from each session, which accounts for the chunk accesses outside the current trace coverage in future sessions.

Table 3.3: Comparison to VMTorrent Approach

Virtual Appliance	System	Workload Duration (sec)	Accessed State (MB)	Fetched State (MB)	Total Buffering Time (sec)	Miss Rate (# misses / # accesses)
Arcanum	vTube	920	87	131	39	0.40%
	VMTorrent	924	84	230	12	0.89%
Riven	vTube	1223	379	566	255	0.70%
	VMTorrent	1222	222	1051	980	0.02%
HoMM4	vTube	926	256	360	136	1.96%
	VMTorrent	927	262	384	54	1.31%

3.4.6 Comparison to VM Distribution Networks

`vTube` employs a VM streaming model with a central repository of virtual appliances. An alternative approach to their dissemination is VM distribution networks, which is based on wide deployment of services in the network that cooperatively deliver the virtual appliance images. Optimized image distribution methods have been proposed, and a notable example is that of VMTorrent [96, 97]. We, therefore, compared the efficiency of `vTube`'s dynamic streaming algorithm and the VMTorrent approach. In order to evaluate the latter approach, we implemented a modified version of the streaming algorithm in `vTube`; following the manner in which VMTorrent distributes VM state, it considers all the chunks that appear in at least one of the traces. These chunks are ordered by their average access time since the start of the session, and the buffering decision is made according to the unmodified dynamic streaming algorithm. Note that this version does not use the notion of clusters, and nor does it dynamically act on the state access by the current VM execution.

Table 3.3 compares the statistics of the two methods for Arcanum, Riven, and HoMM4 on our emulated network with 7.2 Mbps bandwidth and 120 ms RTT. The reported numbers are the average of three measurement runs. Optimizations that are orthogonal to the comparison, such as chunk compression and deduplication, are left intact in both cases. In the measurements, we kept the session duration roughly constant between the two cases: 15 minutes for Arcanum and HoMM4, and 20 minutes for Riven. Overall, the efficiency of state streaming is much higher with `vTube`. For example, with Arcanum, the amount of accessed state during the session is similar, the VMTorrent method fetches 1.8 times more state than `vTube` does. Moreover, miss rate is considerably lower with `vTube` because of its more timely delivery of chunks. The total buffering time of the VMTorrent

method, which is shorter than that of vTube, is a result of less accurate chunk delivery timings. The chunks missed by its buffering period lead to the high miss rate. The impact of coarse-grained streaming by the VMTorrent method is particularly significant with Riven, for which the traces exhibit high variability. It makes the total buffering time as long as 980 seconds out of the total duration of 1,222 seconds. Although this reduces miss rate, the time until VM launch consumes over 80% of the whole session. In contrast, vTube limits the launch time to 255 seconds and allows longer execution of the application, which also resulted in more accessed state. In summary, vTube's algorithm, based on fine-grained state access analysis and its dynamic nature, makes it more efficient compared to the VMTorrent method.

3.5 Summary

Efficient VM delivery to end users over WANs faces the challenge of the sheer VM state size. With tens of gigabytes of VM state, naive approaches to VM transfer over low-bandwidth, high-latency networks can easily take tens of minutes to hours. Such time constraints discourage timely access to virtual appliances as containers of software and data with preserved executability. In this chapter, we introduced our system called vTube, which achieves efficient streaming of virtual appliances over WANs. Leveraging past execution knowledge of VM instances, it performs fine-grained analysis of VM state accesses and accumulates access pattern knowledge for each virtual appliance. In each user session, it applies this knowledge extracted from past execution instances, and streams virtual appliance state while dynamically adjusting to the current VM execution. vTube's algorithm enables rapid launch of virtual appliances within several minutes, even on clients across limited networks such as 3G, 4G LTE, and public Wi-Fi's.

We demonstrated the efficacy of our approach through experiments that revealed the overall behavior, application-level performance, and systems-level statistics of VM execution. Chapter 4 further builds on this work, and investigates an important aspect outside the scope of these types of evaluation: user experience of VM streaming and comparison to remote VM access.

Chapter 4

Interactivity Evaluation of Delivered VMs

We have demonstrated that VM streaming by `vTube` achieves the rapid delivery of virtual appliances over WAN-quality bandwidth. The use of past execution knowledge enables its efficient streaming algorithm, which adopts the video streaming paradigm. In this chapter, we evaluate the effectiveness of this streaming model in aspects not captured by the experiment results previously presented. Specifically, we further investigate the usability aspect of the system directly perceived by users.

During user sessions, `vTube` introduces buffering events to ensure the smooth execution of the VM. When operating over WANs, which often have not only low bandwidth but also high round-trip latency, this model effectively absorbs the impact of the long latency between the server and client by transferring VM state in batches. These properties raise two questions. The first is how buffering events and occasional VM stalls affect the perception of system performance by human users. The second is to what extent the resilience to high latency makes VM streaming practical. In particular, the second question also motivates comparison of VM streaming against remote VM access, a common alternative approach to accessing software encapsulated in VMs. As opposed to VM streaming, which employs a *thick-client* model and executes the VM locally with incremental state delivery, remote VM access uses a *thin-client* model with the VM executing in the cloud. Remote VM access typically does not depend on bandwidth, but instead heavily on the round-trip latency for delivering good user experience. Therefore, the bandwidth and latency characteristics of the user's connectivity to the server determines the practicality of these solutions with respect to each other.

In order to answer the above questions, we conduct human-centric evaluation of VM

streaming and remote VM access. Through a series of user studies comparing the two approaches under varied network conditions, we examine the implications of their computing models on system usability. The results include performance ratings and qualitative evaluation by the participants of the studies. They corroborate the quantitative performance results in the last chapter, such as buffering duration and state miss rates, by showing the actual perception of `vTube`'s streaming capability from the user's perspective.

The work presented in this chapter is a collaboration with Brandon Taylor, doctoral student in the Human-Computer Interaction Institute at Carnegie Mellon University.

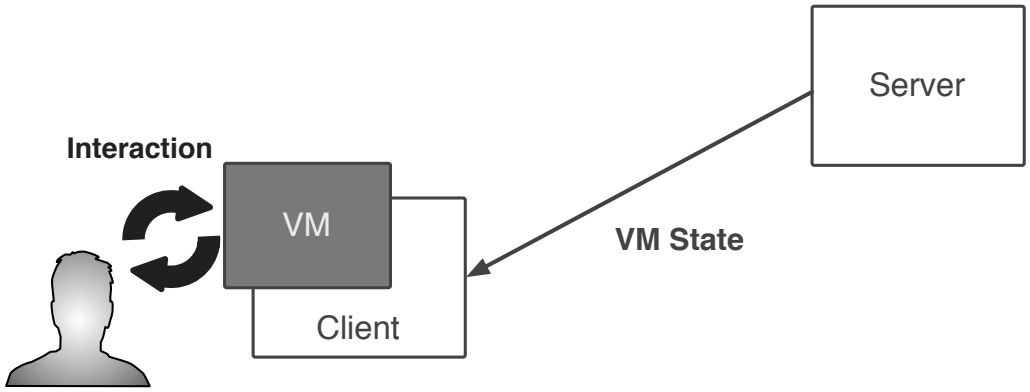
4.1 Properties of VM Access Methods

We conduct further evaluation of VM streaming for two specific purposes. First, one of the two main performance metrics, usability from the user's perspective, eludes systems-level evaluation presented in the previous chapter. In particular, we study the implications of 1) the video streaming paradigm employed by `vTube` with resulting wait time during sessions, and 2) how network conditions affect the perceived performance of the system. Second, we compare VM streaming to an existing alternative approach to VM delivery, remote VM access, in order to evaluate the effectiveness of the streaming model against a latency-sensitive approach.

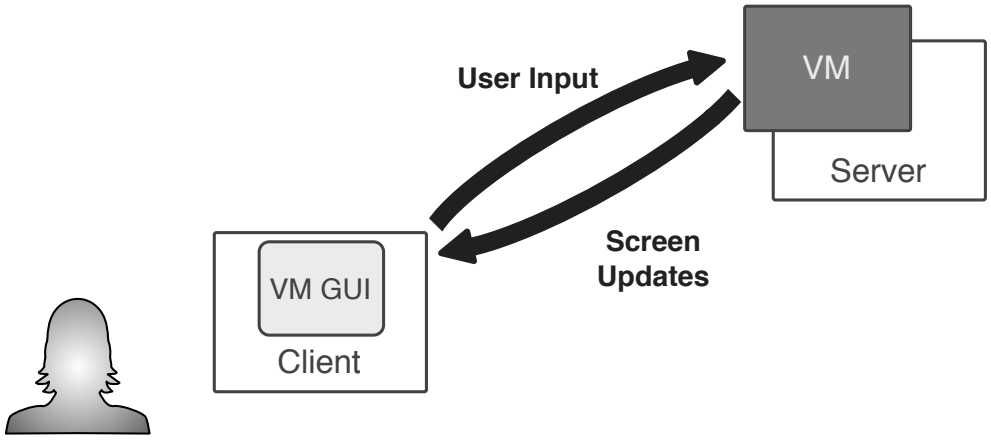
4.1.1 VM Streaming and Remote VM Access

The VM streaming model of `vTube` employs a thick-client model. The client machine performs all the computation on behalf of the VM, while the server is responsible solely for VM state transfer. Figure 4.1 (a) illustrates this computing model. As the VM executes locally, all the user interaction is also handled on the client machine. The VM directly receives input from the user, such as mouse movements and key strokes, processes it, generates output, and presents it on the client's display. Using this model, VM streaming caters to the use cases described in Chapter 3; applications are delivered for local execution in the end user's environment, in some cases accessing data that resides locally.

A major alternative approach to accessing applications encapsulated in VMs is remote VM access using a thin-client model, such as VNC [98]. In contrast to the thick-client model of VM streaming, all the computation for the VM execution occurs on the server side, while the client provides the user with the capability to interact with the VM. This model is depicted in Figure 4.1 (b). The client receives input from the user, and delivers it



(a) VM Streaming



(b) Remote VM Access

Figure 4.1: Computing Models of VM Streaming and Remote VM Access

to the VM running on the server over the network. It also receives screen updates for the VM from the server, and displays them on the local screen. The sole functionality of the client is thus coordinating the interaction between the user and the remote VM, without performing computation for the VM workload itself.

VM streaming and remote VM access both allow the use of computational capability encapsulated in a VM, which are instantiated dynamically by the user. They, however, provide this capability in distinct manners. VM streaming distributes it to the locations of end users. Remote VM access, in contrast, centralizes it and enables end users to make use of computational resources of the server machine. Historically, such thin-client models originate from time-sharing systems, in which users share powerful non-commodity machines with only minimal, and therefore economical, terminals needed on the client side for multiplexing the centrally managed computing resources. Besides the site at which computation is carried out, this distinction leads differences in the network factors that affect the performance of the two approaches.

4.1.2 Impact of Bandwidth and Round-Trip Latency

Two aspects of the network between the server and client machines play a major role in determining the usability of systems using VM streaming or remote VM access: bandwidth and round-trip latency. Bandwidth is a primary performance factor for VM streaming, and round-trip latency for remote VM access.

The VM streaming algorithm buffers state while the VM is suspended, so that its execution is of adequate quality once started. Since VM state is transferred to the client in batches, the duration of the VM suspension depends on the available bandwidth. While remote VM access does not involve such a kind of disruption in VM execution, it instead requires communication between the client and server machines for each user interaction. Specifically, sending user input to and receiving its results from the server incurs at least the round-trip latency between the two machines. For example, when the user moves the mouse cursor, information about the movement is sent to the VM on the server. The VM processes the input, emits graphical updates reflecting the cursor movement, and delivers them to the client. The client then applies the updates to the VM screen presented to the user. VM streaming absorbs this delay per user action and aggregates them efficiently into the buffering periods. Using the same example, when the mouse cursor is moved, the server sends to the client the VM state with which the VM can process the cursor movement. Further cursor movements that follow are likely processed without needing extra state, unless they result in the use of functionalities that have not been executed before. As we will explain further in Section 4.2, remote VM access trades the interaction delays for

immediate accessibility of the VM and relatively low demands for bandwidth. In environments with end users across WANs, we cannot expect bandwidth and round-trip latency to be optimal. The efficiency of VM streaming and remote VM access manifesting in these distinct forms motivates their direct comparison under varied network conditions; it will provide insights into the applicability and relative desirability of the approaches under limited network connectivity.

4.1.3 Goals of Interactivity Evaluation

To summarize, our interactivity evaluation has the following goals in assessing the system usability:

- Conduct human-centric, qualitative evaluation of `vTube` performance, including the efficacy of the VM streaming model.
- Comparison to a common alternative approach, remote VM access, which uses a contrasting model.

We evaluate these aspects in an effective manner by testing varied network conditions and different types of user interaction with the system. Varying these experiment parameters allows us to exercise the systems for VM streaming and remote VM access in multiple ways, thereby revealing the characteristics of their behavior in our target contexts of VM delivery. The design of our methodology, discussed next, explains how we use these parameters to meet our goals.

4.2 Design of Interactivity Evaluation

In order to evaluate the interactive performance of VM streaming and remote VM access, we take an approach of user studies in a managed network environment. Using this approach, we conduct human-centric evaluation of the two techniques under varied network conditions. This method allows us to have the performance of VM streaming and remote VM access rated as perceived by human subjects. Furthermore, by carefully managing the network conditions in which the systems are used, we aim to answer the question of the bandwidth and round-trip latency impacts on the usability of these systems. In the rest of this section, we describe the design and methodology of our studies.

4.2.1 Design Variables

The main challenge in designing the studies is to have a well-defined scope of variables. There exist two main variables in our studies. First, we vary the network condition under which the access to the VM occurs. Second, we test various types of applications. From the administrative perspective, we need to keep the parameter space from expanding excessively, without severely restricting the usefulness of the results.

Network Conditions

We manipulate two parameters of network conditions, bandwidth and round-trip latency between the client and server machines. Considering the factors that primarily affect the performance of VM streaming and remote VM access, we reduce the two-dimensional space of network parameters to one parameter for each of the approaches.

VM streaming introduces buffering periods, whose duration is a function of bandwidth. To the user, these periods are wait time during which the interaction with the VM is prevented. In contrast, the usability of the system is affected by round-trip latency to a lesser extent that is practically negligible. Table 4.1 shows statistics of buffering events and state misses with `vTube` under varied network bandwidth and round-trip latency. The numbers are obtained from test traces with 3 to 4 minutes of Microsoft Word use, during which approximately 490 MB of VM state is delivered to the client. Initial buffering and additional buffering indicate, respectively, the duration of the first buffering event and the total duration of all the other buffering events. Memory and disk misses represent the fractions of demand-fetch requests in all the memory accesses and disk accesses, respectively. Comparison between the two 14.4 Mbps cases indicates that 240 ms and 30 ms round-trip latencies lead to only minor differences in buffering events and state misses. On the other hand, the change in bandwidth from 14.4 to 7.2 Mbps, under the same round-trip latency of 240 ms, results in increases in initial buffering and memory misses. Overall, the round-trip latency contributes to the disruption of VM execution orders of magnitude less than bandwidth does. Based on these observations, we use bandwidth as the parameter to be varied with VM streaming.

Remote VM access, unlike VM streaming, is more sensitive to round-trip latency while it consumes a relatively low bandwidth under most circumstances. In particular, interactive applications that generates graphical output in response to user input operates at a pace dictated by the delay in receiving the input, rather than being constrained by the bandwidth. The cost of propagating input and output becomes more significant as the network latency increases, further impacting the smoothness of the user interaction. For this reason, we

Table 4.1: VM Streaming Statistics under Varied Network Conditions

Bandwidth Latency	Event	Statistics
14.4 Mbps 240 ms	Initial Buffering	135.1 sec.
	Additional Buffering	3.0 sec.
	Memory Misses	0.08%
	Disk Misses	0.03%
14.4 Mbps 30 ms	Initial Buffering	134.2 sec.
	Additional Buffering	0.84 sec.
	Memory Misses	0.02%
	Disk Misses	0.03%
7.2 Mbps 240 ms	Initial Buffering	276.6 sec.
	Additional Buffering	3.4 sec.
	Memory Misses	0.22%
	Disk Misses	0.03%

select the round-trip latency between the local and remote machine as the parameter for remote VM access.

By thus reducing the network parameter space for VM streaming and remote VM access, we keep the number of different system conditions tested to a reasonable range that can be covered by each participant. Also, provided the insignificant impacts of round-trip latency and bandwidth on VM streaming and remote VM access, respectively, we are still able to infer the relative performance of these approaches in ranges of network conditions other than the discrete points we test in the studies. For example, if VM streaming receives a better performance rating than remote VM access for a certain combination of bandwidth and round-trip latency, the former is expected to provide better experience when the bandwidth is higher and the latency remains the same; VM streaming would have shorter buffering events, but remote VM access would not benefit from the higher bandwidth. In this manner, we aim to infer the general trends of their usability under network conditions spanning ranges that are typically expected by end users on the Internet.

Applications

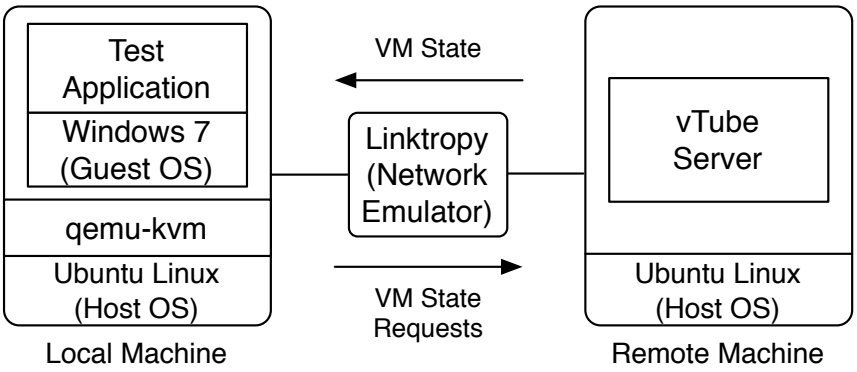
We need to use applications that meet the goal of evaluating interactivity, while having desirable features for user studies. We use the following criteria for selecting multiple applications:

1. Test various types of interaction and investigate how they relate to the usability under different network conditions.
2. Have tasks with variety, rather than the exact same tasks, to mitigate the effect of learning effects obscuring the results.
3. Have tasks that can be completed in a reasonable amount of time.
4. Use applications with fairly large size, so that they are not too trivial for VM streaming to deliver to the client.

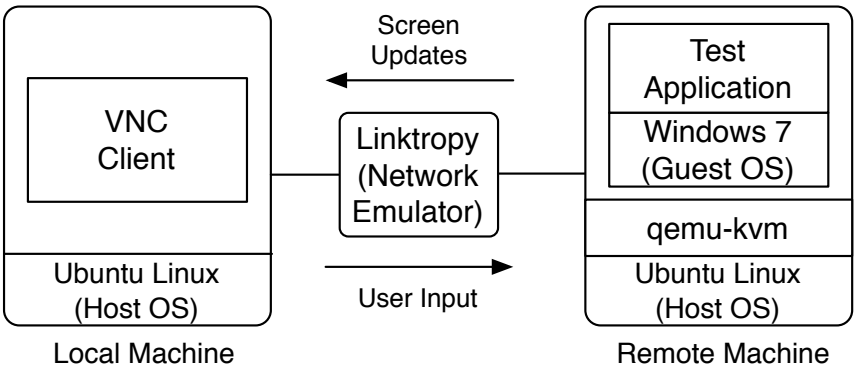
Based on these requirements, we select three types of applications: document editing with Microsoft Word 2007, graphics editing with Adobe Photoshop CS6, and gaming with Bejeweled Twist. The nature of interaction varies adequately among these applications, and tasks with them can be reasonably complex to have enough variety but without excessive difficulty. We describe in Section 4.2.5 how we construct the user tasks for these applications. Also, the VM state size accessed when using them is hundreds of MB or more, and requires a few minutes of buffering with VM streaming.

4.2.2 System Set-Up

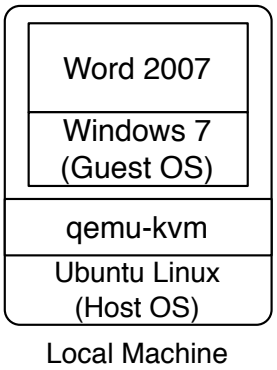
Figure 4.2 describes the system set-up for three configurations used in the studies. We use the `vTube` implementation for VM streaming, and VNC as a representative approach to remote VM access. In the VM streaming case, the `vTube` server runs on the remote machine and streams VMs containing the test applications. The local machine has the client implemented in qemu-kvm, and executes the test application hosted inside a VM. In the VNC case, the remote machine runs the entire stack of software including qemu-kvm and the test application in a VM. The local machine runs a VNC client that connects to the VM and presents its graphical screen to the user. In both of these cases, the network bandwidth and round-trip latency between the remote and local machines are controlled by a dedicated-hardware network emulator. Finally, we use a setting with locally executing VMs as baselines that provides the study participants with a sense of optimal application performance. In this setting, there exists only the local machine, and it executes the VM



(a) VM Streaming (Thick Client)



(b) VNC (Thin Client)



(c) Local Execution

Figure 4.2: System Set-Up for User Studies

Table 4.2: Summary of Test Configurations

Application	Microsoft Word 2007	Adobe Photoshop CS6	Bejeweled Twist
Task	11-Step Instructions	Tutorials	Continuous Gameplay
VM Streaming Bandwidth	7.2, 14.4 Mbps	14.4, 28.8 Mbps	7.2, 14.4 Mbps
Latency	30, 60, 120, 240 ms		
Trial Time Limit	10 Minutes	7 Minutes	7 Minutes
Number of Participants	36	12	12

and handles all the interaction with the user locally. All the machines used are equipped with an Intel Core i7-3770 CPU at 3.4 GHz and 32 GB of memory, running Ubuntu Linux 12.04.

Data Collection

We instrumented qemu-kvm to collect various traces that allow further analysis after the user study trials. First, user input including mouse button presses, cursor movements, and key strokes are recorded. Records of these events provide insights into the user's interaction; for example, key strokes can tell how frequently the user uses short-cut keys, which are more common to those with more experience with the tested application. Second, VM streaming activities by vTube are collected, including buffering events and VM stalls due to state misses. In particular, the buffering events directly correspond to the wait time for the user, which is a primary factor deciding the usability of the system. Third, screen shots of the VM window are captured at 10-second intervals. They help us analyze user behavior that has led to unexpected characteristics in the other traces. The instrumentation for each tracing is made lightweight enough to avoid interfering with the VM execution and affecting the user experience.

4.2.3 Test Configurations

Table 4.2 summarizes the configurations we use in the studies. We describe below the network conditions, trial time, and participant pool for each study per application.

Network Bandwidth and Latency

We vary the network bandwidth and round-trip latency between the remote and local machines in a range that reflects our VM access contexts. Given the distinct performance factors of VM streaming and remote VM access, we test the former under varied bandwidth and the latter under fixed, unrestricted bandwidth. For VM streaming, we select 7.2 Mbps as baseline bandwidth, which is the nation-wide average [18, 75] also used in Chapter 3. With Word and Bejeweled, we also test twice as much bandwidth of 14.4 Mbps. We use 14.4 and 28.8 Mbps for Photoshop for an administrative reason; as the size of the application is larger than the other two, higher bandwidth is necessary for the buffering in VM streaming to complete reasonably within the time allotted in the studies. For remote VM access, we consistently use 100 Mbps. VNC typically requires low bandwidth, in the order of a few Mbps, except for rare cases in which it consume higher bandwidth for significant graphical updates. In order to avoid a bias in favor of VM streaming, we eliminate the impact of such occasional high demands on the user experience by fixing the bandwidth. We test four round-trip latency values: 30, 60, 120, and 240 ms. This range targets round-trip latency expected in environments with various levels of connectivity, from wired WANs to wireless connections including cellular data networks. With these network configurations, we have a total of 7 test configurations per participant and application: two cases for VM streaming with varied bandwidth, four cases for remote VM access with varied round-trip latency, and one case for local VM execution.

Trial Time and Participant Pools

We recruited study participants through an online registration service for research studies. For each study with one of the three applications, we requested a participant pool with some level of familiarity with the application. We first conducted a study using Word with 36 participants, with a 10-minute time limit per test condition. The time limit ensures that the participants can complete all the test conditions within a reasonable amount of time. The participants proceed to the next trial when the time limit is reached, and they do not need to have finished the task by that time. Based on the consistency we observed in the results of the Word study, we continued with the Photoshop and Bejeweled studies at a reduced scale, each with 12 participants and a 7-minute time limit per trial. Given these time-managed trials, the participants completed the study in approximately 1 hour without being considerably affected by fatigue. Demographic information about the participants was collected on a voluntary basis; the distributions of their ages, which we were able to receive from all of them, are summarized in Table 4.3. The participants were recruited per study for each application, with no overlapping between studies. The studies with these

Table 4.3: Age Distributions of Study Participants. The table shows the number of participants in certain ranges of age per study for each application.

	Age Range					Total
	18-24	25-34	35-44	45-54	55+	
Word	26	9	0	1	0	36
Photoshop	8	2	1	1	0	12
Bejeweled	5	5	1	0	1	12

participants were approved by Carnegie Mellon University, under IRB Protocol Number HS13-668.

4.2.4 Procedure

The studies are carried out in an in-lab setting on the Carnegie Mellon University campus. One participant is tested at a time, while one or more administrators are on site for assistance and monitoring. The study procedure with each participant proceeds in the following manner. First, we describe the purpose of the study to the participant upon arrival. The participant also completes a short survey that asks questions about experience with computing in general, and that with the particular application being tested. Then, the participant starts the seven trials. The local VM execution case is used as the first trial, which provides ideal, baseline performance against which the participant can compare the user experience of the following trials. Once done with the first trial, the participant fills out a short survey about the user experience during the trial. The participant is asked to consider all the factors and events during the trial, in particular any wait time, freeze, and stalls of the application. After the survey, the remaining six trials with different network conditions and VM access methods are conducted, each followed by a survey in the same manner. The study completes after all the seven trials are done.

Per-Trial Survey

The survey after each trial consists of questions that ask for evaluation of the user experience in multiple aspects. They use a subset of NASA-TLX dimensions [11], in which the evaluation is done on a 5-point Likert scale: Very High, High, Medium, Low, and Very Low. We constructed four questions regarding the participant's 1) sense of satisfaction, 2)

feelings of annoyance, 3) mental demands, and 4) sense of accomplishment, in performing the assigned task using the system. These questions are presented in the survey as follows:

- How would you rate your satisfaction with the system performance during this trial?
- To what degree did you experience at least one of these feelings: insecure, discouraged, irritated, stressed, or annoyed?
- How mentally demanding was the task during this trial?
- How well were you able to accomplish the task?

We use the ratings given as the answers to these questions in order to quantize the qualitative perception of the system usability by the participants. Additionally, the survey includes two open-ended questions:

- Was there any aspect of the system performance that stood out in your mind? Please explain.
- Was there any aspect of the instructed task that stood out in your mind? Please explain.

These questions are intended as further feedback on the user experience during the trial, thereby providing further insights into the ratings given as the responses to Likert-scale questions.

4.2.5 Test Applications

The three interactive applications we test have different manners in which the user interacts with, and expects results from, the program. Some of them, for example, rely more on mouse movements and require highly synchronized graphical updates, while others involve key strokes and can tolerate a certain degree of delays in interaction without significantly sacrificing the usability of the program.

Word Document Editing

Our first application is Microsoft Word 2007 [9]. The user edits a document in various manners following instructions. We created seven instruction sets to be used, one for each

test condition; the paring between the instructions and conditions are varied the across participants, in order to avoid biases in the study results. One set of instructions consists of 11 individual steps, each of which asks for the use of a distinct functionality of the program. For example, a step reads, "Use the Find function to find the first instance of the word 'present' ...," or "Use the Text Box function to create a caption labeling the picture as ..." The types of functionalities used are as follows:

- Opening documents
- Copying and pasting contents
- Font formatting
- Page formatting
- Creating a table
- Counting the number of words
- Searching a particular word
- Deleting and inserting a picture
- Inserting a text box
- Saving the document

The instruction sets use unique documents, with the above functionalities appearing in different orders except for certain steps, such as adding a caption immediately after inserting an image and saving the document at the end.

The tasks are completed through combinations of key typing and mouse input. Depending on the skill level, one user may accomplish the same step faster than another by using a more efficient method. For example, there are a number of short-cut keys for common functionalities of Word, which are otherwise done through graphical interaction such as dialog boxes. Also, tasks have varying implications on how a VM stall is perceived by the user. The user can likely continue typing a few words without seeing them appear on the screen one by one. When scrolling within the document, on the other hand, the user needs to wait for the screen to properly update in order to know the current position in the document.

Photoshop Tutorials

Our second application is Adobe Photoshop CS6 [14]. The participants are provided with tutorials as instructions, and perform tasks following them on the application. Given the level of familiarity with Photoshop that can be expected from the participant pool, we selected to use tutorials instead of regular editing tasks.. As done in the Word case, we created 7 tutorials to be used for the 7 test conditions, with varied pairing across the participants. Each tutorial walks the user through a number of steps that teach how to use a particular functionality of Photoshop. The tutorials consist of the contents in the following list:

- Layers, including their creation, selection, and alignment.
- Free Transform, which can arbitrarily modify the shape of a figure.
- Warp Text, which allows shaping text into various forms.
- Custom Shape, with which the form of various illustrations can be manipulated.
- Magic Wand, which extracts a color from a picture and applies it to other functionalities.
- Color Range, which allows specifying a color by parameter values.
- Photo Effects, which applies various useful effects on pictures in a simplified manner.

These tutorials mainly require mouse interaction, with infrequent key entries. Shortcut keys for certain functionalities are also available. The tasks involve graphical updates, often spanning a large portion of the screen. Applying a filter to an image displayed on the screen, for example, changes most of its pixels. This type of user workload, therefore, generates guest screen updates of considerable size, and can render the responsiveness of the system sluggish when using VNC.

Bejeweled Twist Gameplay

Our third application is a puzzle game called Bejeweled Twist [15]. In this game, the user rotates positions of various jewels on a grid, and aligns three of them with the same color to make them disappear and gain points. Unlike the other test applications, we do not use instructions and instead have the user play the game to achieve as high a score as possible.

If the game ends in the middle of the trial, the score is saved and the user starts a new game to continue playing.

The user interaction in this game relies on mouse movements and clicks almost exclusively, rarely needing keyboard input. The graphical output uses frequent animation when, for example, jewels move or disappear on the grid. VM stalls cause delays in the motions, and render themselves easily noticeable. These delays, however, do not necessarily interfere with the user actions as might be expected. The gameplay progresses with user actions and resulting animation happening in an alternating fashion. Thus, after an action, the user naturally waits briefly until its result is reflected on the screen, instead of being interrupted from performing continuous actions by delayed responses from the application.

4.3 Results

Following the types of questions in the surveys, the results of the studies have two categories: performance ratings and qualitative observations. We first present the ratings of user satisfaction, feelings of annoyance, mental demands, and sense of accomplishment, collected from the Likert-scale questions. Next, we discuss insights into the relation between user experience and the behavior of the application and system, which are derived from the two open-ended questions.

4.3.1 User Satisfaction

Figure 4.3 and Table 4.4 show the ratings of satisfaction with the system performance across the applications and test conditions. The vertical bars represent confidence intervals, to which 95% of the corresponding rating values belong. The local execution case has good ratings around High, as expected. VNC with 30 ms latency has ratings close to this baseline, between High and Medium. As the latency increases, the ratings become worse; Word is the most sensitive to latency, reaching Medium already at 60 ms and eventually becoming Low at 240 ms. The ratings for Photoshop and Bejeweled show less severe degradation than Word, staying slightly below Medium. Note that although Bejeweled has a better rating at 60 ms than at 30 ms, the difference is not statistically significant as indicated by the confidence intervals. VM streaming of Word and Bejeweled yields ratings slightly below Medium with 7.2 Mbps. At 14.4 Mbps, it shows a greater improvement with Word than with Bejeweled, staying above Medium. Photoshop at 14.4 Mbps has a rating similar to those of Word and Bejeweled at 7.2 Mbps. Its rating at 28.8 Mbps, also, is comparable to the Word rating at 14.4 Mbps. In summary, VNC at 30 ms provides perfor-

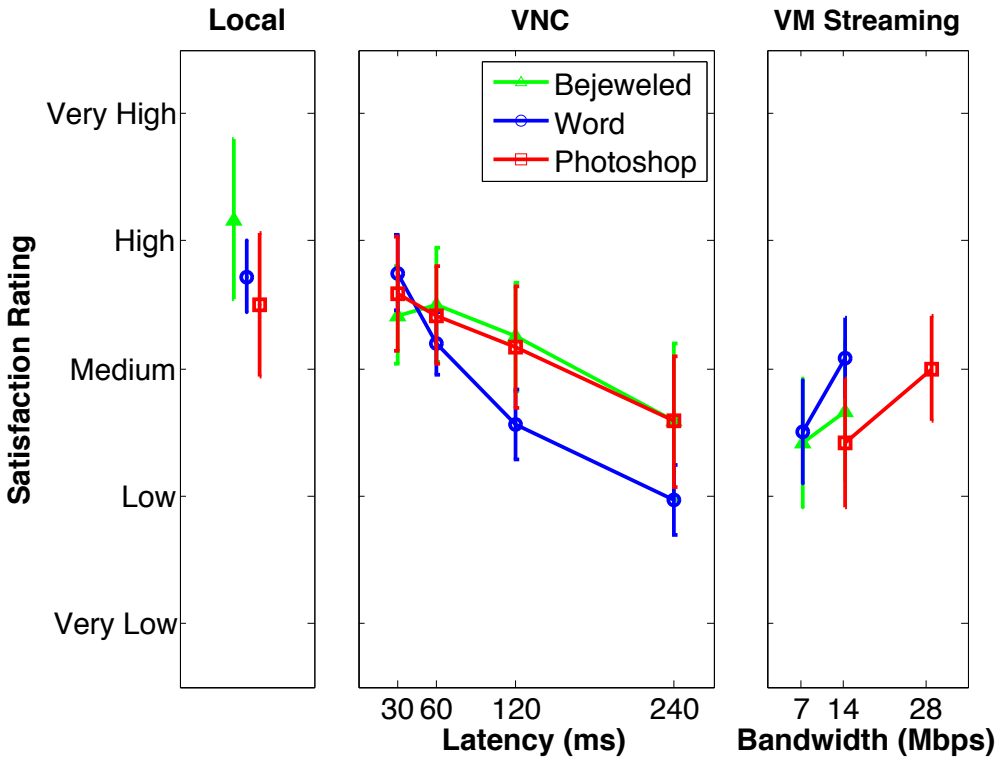


Figure 4.3: Ratings of Satisfaction with System Performance. The ratings are averaged per test condition and shown with 95% confidence intervals.

mance similar to the local execution, for all the applications. With Word, VM streaming at 7.2 and 14.4 Mbps is comparable to VNC at 120 and 60 ms, respectively. With Photoshop, the ratings of VM streaming at 14.4 and 28.8 Mbps resemble those of VNC at 240 and 120 ms. With Bejeweled, VM streaming at 7.2 Mbps roughly matches VNC at 240 ms, while it only slightly improves as the bandwidth increases to 14.4 Mbps.

The ranges of network bandwidth and round-trip latency suitable for VM streaming and VNC are visualized in Figure 4.4. VM streaming is expected to have higher user satisfaction than VNC towards the upper right corner, which corresponds to higher bandwidth and higher latency. VNC is more preferable towards the opposite direction, which represents lower bandwidth and lower latency. The cross marks on the lines indicate the points

Table 4.4: Statistics of Satisfaction with System Performance. The numbers are calculated using numerical values with 5 representing Very High and 1 Very Low. STD stands for standard deviation.

		Word		Photoshop		Bejeweled	
		Rating	STD	Rating	STD	Rating	STD
Local Execution		3.72	0.85	3.50	1.00	4.17	1.11
VNC	30 ms	3.75	0.91	3.58	0.79	3.42	0.67
	60 ms	3.19	0.75	3.42	0.67	3.50	0.80
	120 ms	2.56	0.84	3.17	0.83	3.25	0.75
	240 ms	1.97	0.84	2.58	0.90	2.58	1.08
VM Streaming	7.2 Mbps	2.50	1.23	-	-	2.42	0.90
	14.4 Mbps	3.08	0.97	2.42	0.90	2.67	0.98
	28.8 Mbps	-	-	3.00	0.74	-	-

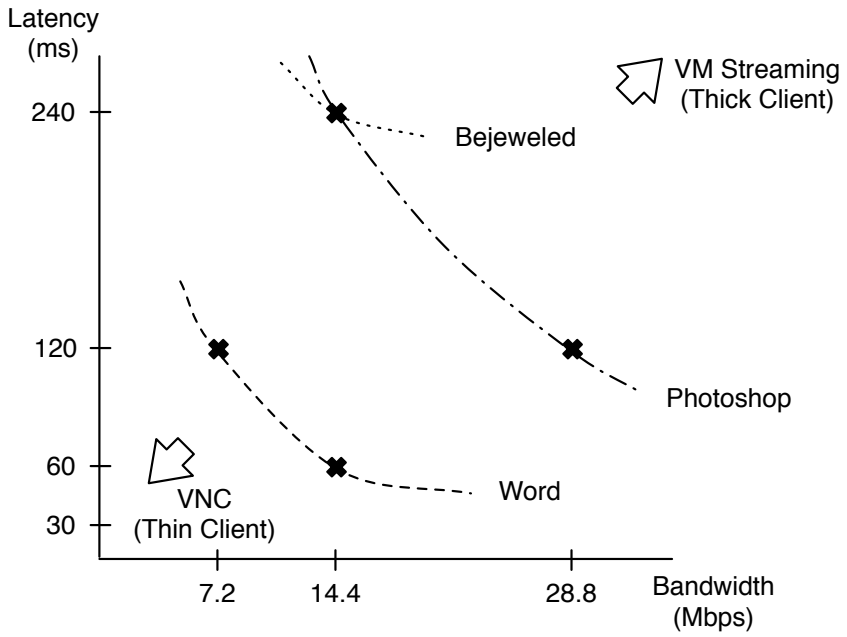


Figure 4.4: Bandwidth-Latency Ranges Suitable for VM Streaming and VNC in Terms of Satisfaction. The cross marks represent points at which ratings are comparable between VM streaming and VNC.

at which the ratings of VM streaming and VNC are comparable in the results shown in Figure 4.3. While the number of data points is limited, the lines are drawn to approximately divide the two-dimensional space into regions for which VM streaming or VNC is expected to provide higher satisfaction. In general, VM streaming is expected to provide higher satisfaction with higher bandwidth, while the impact of latency is insignificant. Among the three applications, Word has the largest region for VM streaming. For Photoshop, the line is moved towards the upper right, requiring higher bandwidth or higher latency for VM streaming to be more preferable than VNC. Lastly, the line for Bejeweled stays higher than the other lines. VNC tolerates higher latency and remains more preferable than VM streaming, unless used under high bandwidth.

4.3.2 Feelings of Annoyance

Figure 4.5 and Table 4.5 show the ratings of feelings of annoyance. The local execution case has the lowest annoyance with Bejeweled, and the rating is higher with Word and Photoshop. For these applications, the nature of the assigned task is different from that for Bejeweled. The participants follow instructions to use particular functionalities, with which they are not necessarily familiar. Since the local execution case is always the first trial for the participants, they are also not accustomed to the procedure and therefore may tend to give a higher rating of annoyance. The participants, on the other hand, tend to learn the rules of Bejeweled quickly or have experience playing the game.

With VNC, the ratings monotonically worsen as the round-trip latency becomes higher. At 30 ms, the ratings are similar to those of the local execution case, with Word and Photoshop being better because of the above reason. Similar to the case of satisfaction ratings, Word is the most sensitive to the increase in latency. Photoshop is the most insensitive, having little difference between 120 and 240 ms. Bejeweled resembles Photoshop up to 120 ms, but its rating becomes close to Medium at 240 ms. With VM streaming, the ratings of Word at 7.2 and 14.4 Mbps are comparable to those of VNC at 120 and 60 ms, respectively. Photoshop has higher annoyance levels than the VNC cases, despite the higher bandwidth range, likely because of the initial buffering time being longer than with the other applications. Interestingly, Bejeweled has higher ratings of annoyance than Word and Photoshop. One factor that causes this phenomenon is that the participants read the instructions while waiting for the VM to start with these two applications. Bejeweled does not have instructions, and thus they are left without any task until the initial buffering completes, finding the wait time more annoying than in the other cases.

Figure 4.6 visualizes the bandwidth-latency ranges in which VM streaming or VNC causes less annoyance than the other. Note that with Photoshop, VM streaming at 28.8

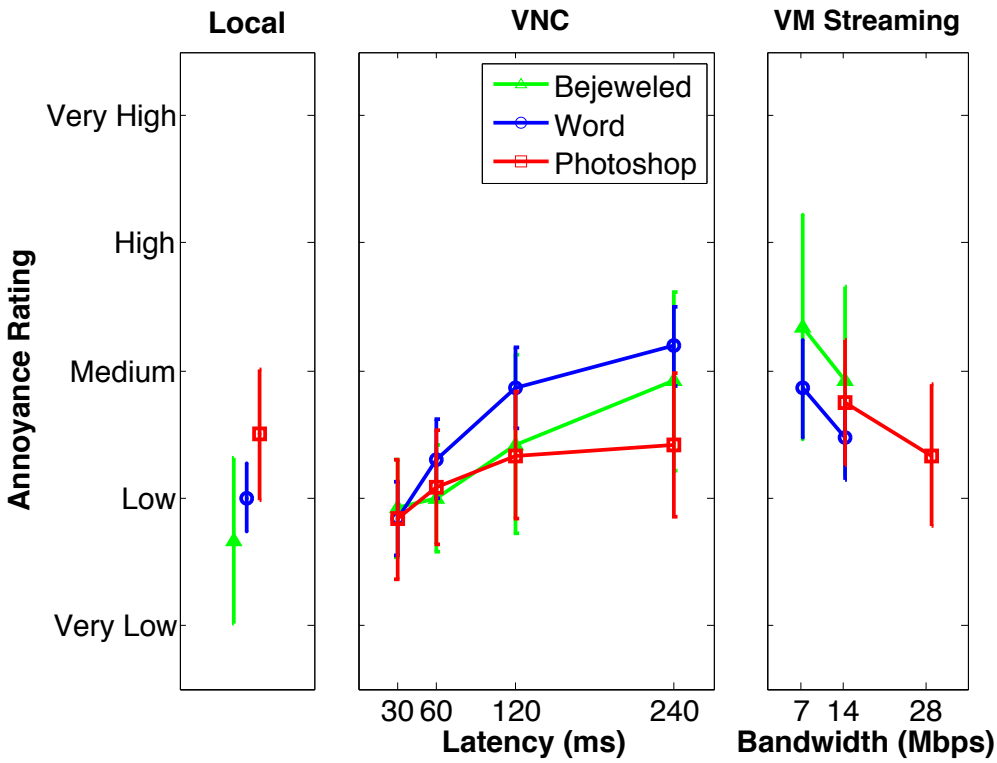


Figure 4.5: Ratings of Feelings of Annoyance. The ratings are averaged per test condition and shown with 95% confidence intervals.

Mbps has a similar rating to those of VNC at 120 and 240 ms. The corresponding curve is accordingly drawn in the figure. This curve is a major difference compared to the satisfaction mappings in Figure 4.4. Specifically, while the participants are more satisfied with VM streaming at 28.8 Mbps than with VNC at 240 ms, they feel almost the same level of annoyance in both of the cases. This result indicates that the wait time for VM state buffering can be a cause of frustration depending on the task, which also relates to the level of task completion achievable in the allotted time. We further discuss the effect of task completion on the system usability later in this section.

Table 4.5: Statistics of Feelings of Annoyance. The numbers are calculated using numerical values with 5 representing Very High and 1 Very Low. STD stands for standard deviation.

		Word		Photoshop		Bejeweled	
		Rating	STD	Rating	STD	Rating	STD
Local Execution		2.00	0.83	2.50	0.90	1.67	1.15
VNC	30 ms	1.83	0.88	1.83	0.83	1.92	0.67
	60 ms	2.31	0.95	2.08	0.79	2.00	0.74
	120 ms	2.86	0.99	2.33	0.89	2.42	1.24
	240 ms	3.19	0.95	2.42	1.00	2.92	1.24
VM Streaming	7.2 Mbps	2.86	1.17	-	-	3.33	1.56
	14.4 Mbps	2.47	1.03	2.75	0.87	2.92	1.31
	28.8 Mbps	-	-	2.33	0.98	-	-

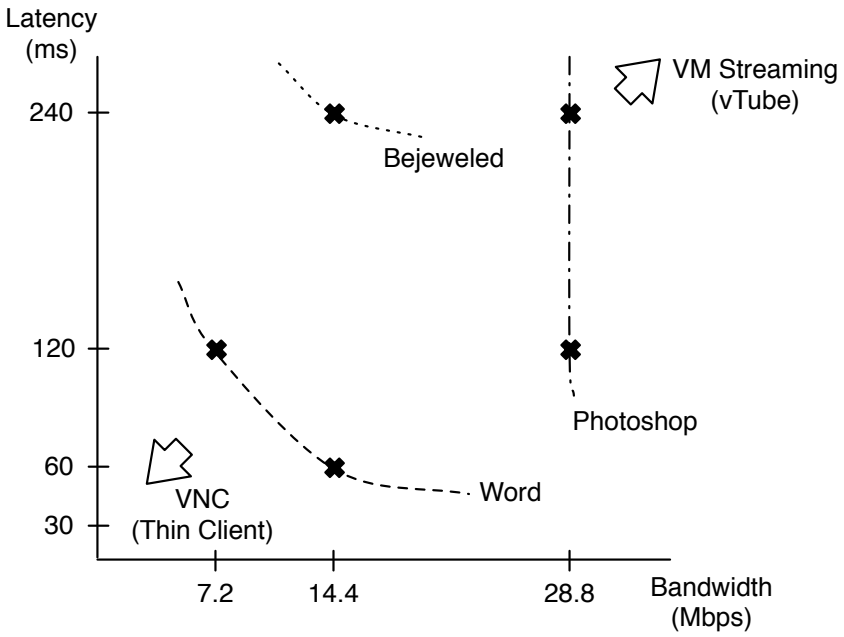


Figure 4.6: Bandwidth-Latency Ranges Suitable for VM Streaming and VNC in Terms of Annoyance. The cross marks represent points at which ratings are comparable between VM streaming and VNC.

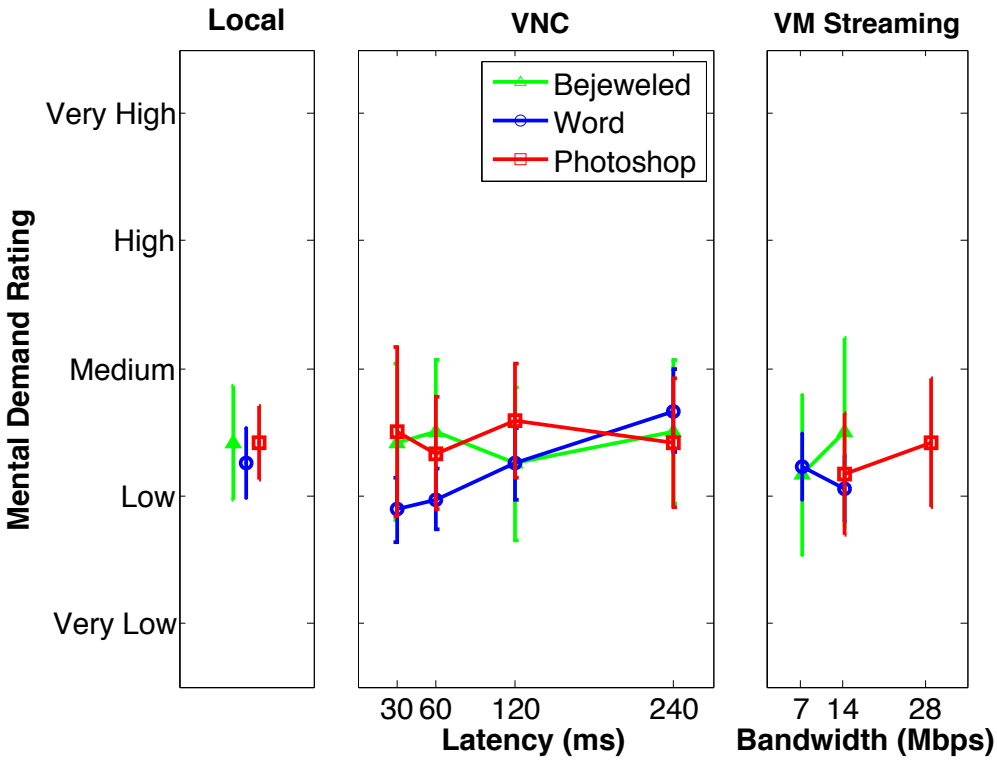


Figure 4.7: Ratings of Mental Demands. The ratings are averaged per test condition and shown with 95% confidence intervals.

4.3.3 Mental Demands

Figure 4.7 and Table 4.6 present the ratings of mental demands. Among the three applications, only Word exhibits a discernible trend; the mental demand increases as the latency becomes higher with VNC, or as the bandwidth decreases with VM streaming. Still, considering the ratings for the local execution case and the confidence intervals, the difference between these ratings are not significantly large. Also, Photoshop and Bejeweled do not have monotonically changing ratings, and thus mental demands are not clearly affected by the network conditions.

Table 4.6: Statistics of Mental Demands. The numbers are calculated using numerical values with 5 representing Very High and 1 Very Low. STD stands for standard deviation.

		Word		Photoshop		Bejeweled	
		Rating	STD	Rating	STD	Rating	STD
Local Execution		2.25	0.84	2.42	0.51	2.42	0.79
VNC	30 ms	1.89	0.78	2.50	1.17	2.42	1.08
	60 ms	1.97	0.74	2.33	0.78	2.50	1.00
	120 ms	2.25	0.87	2.58	0.79	2.25	1.06
	240 ms	2.67	0.99	2.42	0.90	2.50	1.00
VM Streaming	7.2 Mbps	2.22	0.80	-	-	2.17	1.11
	14.4 Mbps	2.06	0.79	2.17	0.83	2.50	1.31
	28.8 Mbps	-	-	2.42	0.90	-	-

4.3.4 Sense of Accomplishment

Figure 4.8 and Table 4.7 show the ratings of sense of accomplishment. In general, the ratings become worse as the bandwidth decreases or the latency increases. We attribute the low accomplishment rating for Photoshop in the local execution case to the level of familiarity with the application; the participants often need to explore the user interface of Photoshop in the first trial, feeling that they have accomplished less compared to the other trials. Bejeweled does not have as clear a trend in the ratings as the other two applications have. It is expected that the instruction-based tasks contribute to clear sense of accomplishment more than the unrestricted gameplay. Compared to the ratings of user satisfaction in Figure 4.3, the ratings of VM streaming fall into a range closer to those of VNC. Although VM streaming has the initial buffering time, once the VM starts the user can expect smooth interaction and good productivity in performing the assigned task. This nature of interaction helps to sustain the sense of accomplishment, while user satisfaction is likely to be more affected by the initial buffering time.

4.3.5 Qualitative Observations

In addition to the four rating results presented above, we collected various qualitative observations from the two open-ended questions in the per-trial survey. Also, we observed a trend in the relationship between ratings and task completion from additional statistics

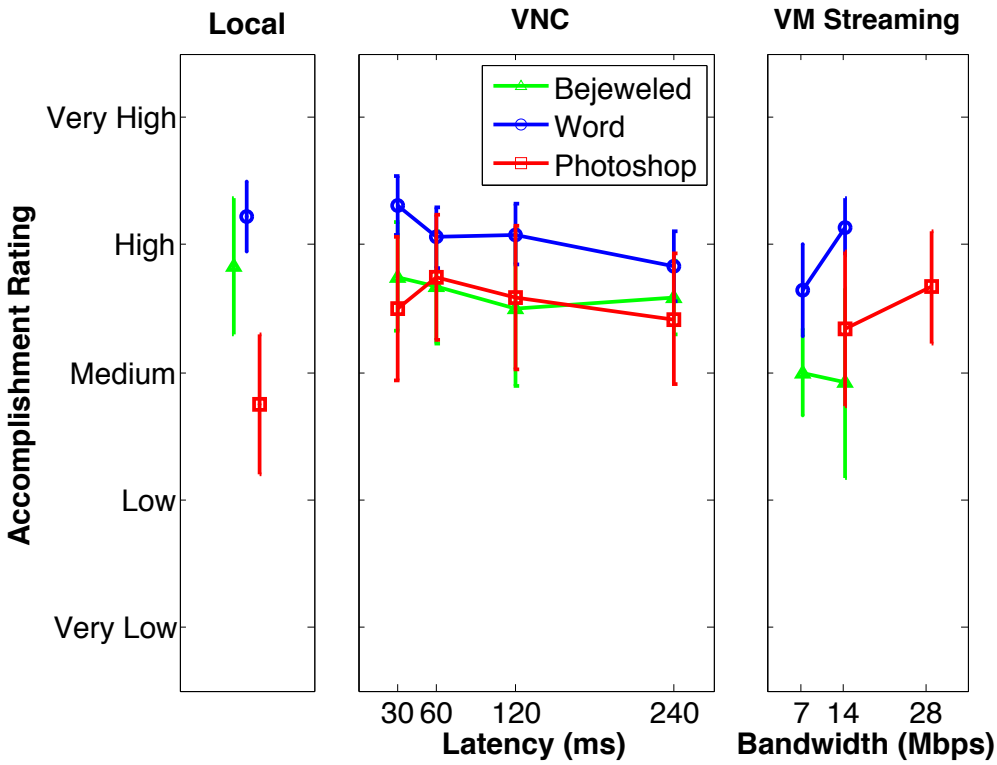


Figure 4.8: Sense of Accomplishment. The ratings are averaged per test condition and shown with 95% confidence intervals.

regarding the participants. These observations led to major findings about the impact of round-trip latency on user interaction under VNC, and about an effect that buffering in VM streaming has on system usability.

Impact of Latency on Interaction

Depending on the type of interaction used by the application, its sensitivity to round-trip latency varies. One notable example is animation, which is extensively used by Bejeweled throughout the game play. As evidenced by the user satisfaction ratings in Figure 4.3, Bejeweled has a clear difference in the performance perceived by the user between the

Table 4.7: Statistics of Sense of Accomplishment. The numbers are calculated using numerical values with 5 representing Very High and 1 Very Low. STD stands for standard deviation.

		Word		Photoshop		Bejeweled	
		Rating	STD	Rating	STD	Rating	STD
Local Execution		4.22	0.83	2.75	0.97	3.83	0.94
VNC	30 ms	4.31	0.71	3.50	1.00	3.75	0.75
	60 ms	4.06	0.71	3.75	0.87	3.67	0.78
	120 ms	4.08	0.73	3.58	1.00	3.50	1.09
	240 ms	3.83	0.85	3.42	0.90	3.58	0.51
VM Streaming	7.2 Mbps	3.64	1.10	-	-	3.00	0.60
	14.4 Mbps	4.14	0.68	3.33	1.07	2.92	1.31
	28.8 Mbps	-	-	3.67	0.78	-	-

local execution case and the VNC case with 30 ms latency. One participant stated, "visual artifacts degraded [the] overall image for the game," and rated the VNC case with 30 ms lower than the local execution.

At the same time, the degree to which the rating worsens as the latency increases is not significantly different from the Photoshop case, whose local execution and VNC at 30 ms cases have similar ratings. One of the participants answered in the survey after the trial with VNC at 240 ms, "[There was] an increased lag between mouse and pointer, with more intense lag coming from animations, but [it] had a very low impact on game play." Similarly, survey answers after Photoshop trials indicate that the application interaction is also relatively insensitive to the increase in latency. One participant noted, for the application, "the mouse cursor kept lagging, but it did not bother me too much." In contrast, the user experience with Word tends to be affected by the increase in latency. For example, a participant was not affected by 30 ms latency, but made a comment about 60 ms latency that "it lagged when I was scrolling," and gave worse ratings for satisfaction and annoyance.

Word differs from Photoshop and Bejeweled in that its task consists of many small actions from which the user typically expects a fast response. For example, many functionalities involve opening and clicking on a drop-down menu, followed by a dialog box. If the user chooses to use a short-cut key, also, a dialog box or other graphical output is expected to appear immediately on the screen. Based on the feedback from the participants, they do not recognize short delays around 30 ms as significant enough divergence

from their expectations. However, as the latency increases, they start to notice the delayed responses and become frustrated. The Photoshop and Bejeweled tasks, on the other hand, include heavier actions with which the user can tolerate delays to some extent. Photoshop involves whole-image manipulation throughout the tutorials, and Bejeweled extensively uses animation in its visual effects. These types of interaction are likely to have clearer separation between the input and output phases in the user's mind.

Cost Amortization by VM Streaming

VM streaming and VNC, as described in Section 4.1, have distinct computing models. VM streaming introduces wait time while buffering VM state, so that user experience is expected to be adequate when the VM executes. VNC allows the user to start interacting with the VM immediately, while the impact of round-trip latency on interactivity persists throughout the user session. This difference yields important implications in our studies. We limit the time for each trial, for administrative reasons, to 7 to 10 minutes. In VM streaming, the participants can start performing the assigned tasks only after the initial buffering is complete. The amount of time they have for the tasks, therefore, is shorter with VM streaming than with VNC. As a result, VM streaming has a lower chance of task completion, which affects user satisfaction.

In the study with Word, 92% of the participants completed the tasks within the time limit with VM streaming at 14.4 Mbps, which has approximately 2 minutes of the initial wait for buffering. At 7.2 Mbps, the wait time increases to 4 minutes and only 53% of the participants were able to complete the tasks. Table 4.8 summarizes the satisfaction and accomplishment ratings for Word with VM streaming, separated into two groups in which the participants complete or do not complete the task. For satisfaction, when the participants do not complete the task, the rating is worse by more than 0.5 points compared to when they complete the task. For accomplishment, the degradation is more considerable and is almost 1.0 points. With VNC, on the other hand, 98% of them completed the Word tasks across all the network conditions. Thus, the level of task completion did not have as much an impact as it did on VM streaming.

As VM streaming amortizes the cost of a buffering event over a period of time, it would likely lead to better sense of satisfaction and accomplishment if there was no time limit for completing the tasks. Ultimately speaking, once all the necessary VM state has been cached on the local machine, user experience reaches the level of the local execution. In contrast, user experience with VNC is not expected to improve over time. Persistent delays due to the latency between the remote and local machines, instead, may have a cumulative negative effect on the perception of the user. If the participants were to use the system

Table 4.8: Ratings of Satisfaction and Accomplishment for Word by Task Completion. The ratings are averaged in each category, using numerical values with 5 representing Very High and 1 Very Low. The numbers in parentheses indicate standard deviation.

	Satisfaction		Accomplishment	
	Complete	Incomplete	Complete	Incomplete
7.2 Mbps	2.84 (1.34)	2.12 (0.99)	4.11 (0.88)	3.12 (1.11)
14.4 Mbps	3.12 (0.99)	2.67 (0.58)	4.21 (0.65)	3.33 (0.58)

under VM streaming and VNC for extended periods of time, the former thus may have better ratings relative to the latter.

4.3.6 Implications on the effectiveness of VM streaming

The above results and findings from the studies validate the effectiveness of the computing model used by `vTube`. As presented in Figure 4.4, VM streaming serve as a competitive solution under ranges of low-bandwidth and high-latency conditions. Our studies were designed to avoid a bias in favor of VM streaming relative to remote VM access; in particular, we set maximum duration for each trial, which limits the benefit of cached VM state. The evaluation by the participants considered the initial buffering as a usability factor, while it affects task completion as noted above. Nonetheless, `vTube` makes the seemingly heavyweight process of VM instantiation across a network a desirable, cost-effective approach especially when high round-trip latency discourages remote VM access. Additional factors in networks, such as jitters in the round-trip latency often emerging in cellular networks, can further complicate and negatively impact the usability of remote VM access. The VM streaming model naturally masks such latency issues, and works in a robust manner as long as the bandwidth remains at a reasonable level.

4.4 Summary

We evaluated the interactive performance of VM streaming by `vTube` through a series of user studies. The studies address two important aspects of its usability: how its stream-

ing model affects the perception of users, and what implications it has on VM delivery over low-bandwidth and high-latency networks. We investigated these questions through comparison to a common alternative approach, remote VM access, using VNC. Ratings of their usability in multiple categories indicate the desirability of these approaches relative to each other, for three distinct types of applications and under varied network bandwidth and latency. The results validate the effectiveness of the VM streaming model, which absorbs the impact of high round-trip latency through the introduction of buffering events. In the ranges of network conditions we tested, round-trip latency of 120 and 240 ms is commonly experienced on 3G and 4G LTE networks, and bandwidth between 7.2 and 14.4 Mbps is typical on these networks and public Wi-Fi's. `vTube` offers a competitive alternative to VNC in these practical WAN environments. In addition, qualitative evaluation by our studies provides further insights into various ways in which interaction delays and wait time affect the usability of the two systems.

Chapter 5

Urgent Transfer of VMs under Contention

Constraints that challenge the timeliness of VM transfer emerge not only as network resources, which we have explored in the previous chapters, but also as the resource capacity of the VM hosts. This latter case is the second target context in this thesis work. Specifically, although a well-established technique for load balancing in clouds, VM migration remains to be a challenging problem under high loads and with ever-growing VM state size. The cost of the migration operation, especially for sustaining the liveness of the VM being migrated, renders its timely completion hard to achieve. This cost, in turn, discourages oversubscription as it may considerably sacrifice performance at peak times. Oversubscription is a way of improving resource efficiency by co-locating VMs on the same host and assigning them physical resources as needed. Server consolidation is a common example that is widely practiced. While oversubscription may sufficiently handle VMs under their average loads, when under high loads they can collectively demand more than the physical resource limit. Therefore, vendors with performance emphasis are forced to relinquish this strategy and use conservative resource allocation, such as Placements on Amazon EC2 [4]. Ideally, administrators should be able to aim for resource efficiency without sacrificing performance. They must otherwise rely on static, conservative resource allocation, wasting some physical resources. This dilemma calls for an approach to migration that urgently transfers VMs under contention to a new host.

There exists a variety of approaches to migration employing different methods of state transfer, live migration [40, 90] being the most widely adopted method. Unfortunately, live migration does not adequately address the aforementioned urgent load balancing for VMs under contention, often taking tens of seconds until freeing computational resources

on the source. Live migration can lead to such elongated duration of the migration process because of its primary focus on minimizing down time. When VMs are under contention, however, we need to resolve their mutual interference and salvage their performance quickly. This demands that the migration operation perform well not exclusively in achieving minimal down time, but also in transferring VM execution to another host. The requirement becomes especially important when the VMs provide services that need to sustain high overall throughput, despite short but non-minimal down time. Such scenarios include migrating back-end or batch-processing applications, and quarantining malfunctioning VMs or those under DoS attacks for diagnosis in segregation.

Besides its focus on the liveness of the target VM, a key limitation of current techniques is that they are fundamentally blackbox approaches. As a result, they compensate for the lack of knowledge by monitoring the workload, extending the duration of migration. This chapter presents *enlightened post-copy* migration, a whitebox approach based on *current execution knowledge* of the guest OS. A whitebox technique, unlike blackbox, uses explicit cooperation between multiple layers in the systems hierarchy. With virtualization being prevalent in today's computing world, we treat migration as a functionality natively supported by the guest OS. The approach is driven by enlightenment [85], a type of knowledge passed by the guest to the hypervisor for improving the efficiency of its operation. In enlightened post-copy, current execution knowledge is passed through enlightenment to the hypervisor; the guest OS informs the hypervisor of memory regions that require high transfer priority for sustaining the guest performance. The hypervisor then migrates the guest state following the passed information in a post-copy manner. It suspends the VM and resumes it immediately on the destination, while continuously pushing the state in the prioritized order and also serving demand-fetch requests by the resumed VM. The use of post-copy, as opposed to common pre-copy, facilitates the guest cooperation and rapidly resolves the interference between the contending VMs.

5.1 Problem and Objectives

Historically, VM migration has focused on the liveness, namely minimal suspension, of the target VM. After describing key criteria in migration first, we examine this common behavior of state-of-the-art migration algorithms and implementations. Reflecting on the observations, we then derive the characteristics desired in our solution to urgently resolving VM contention.

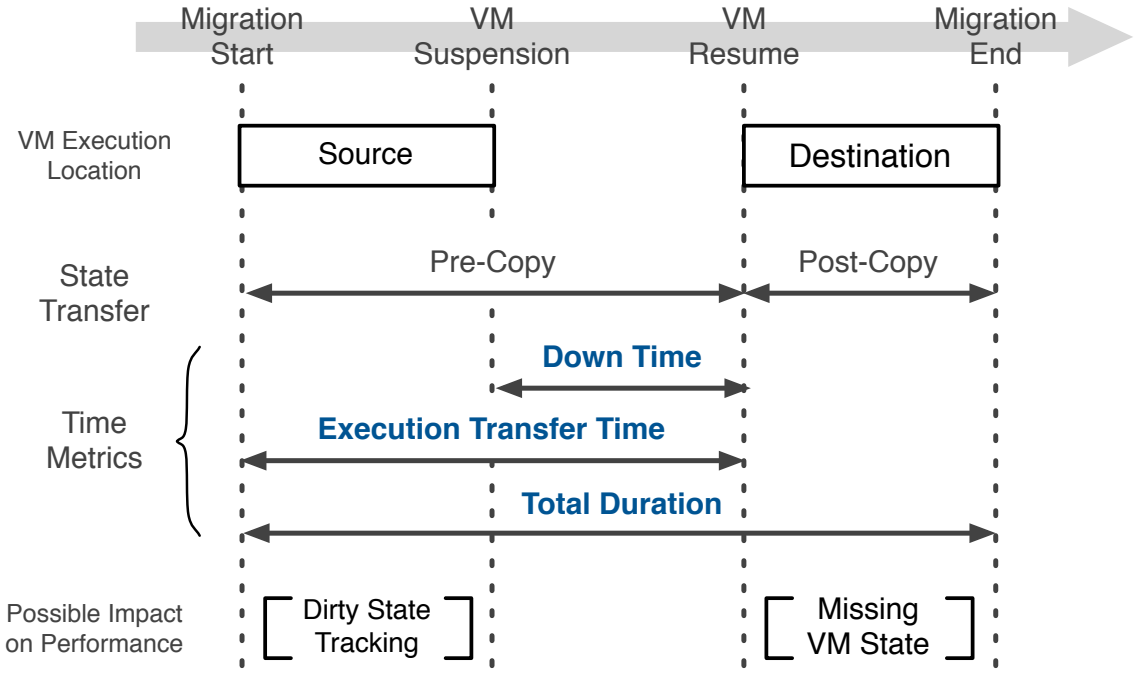


Figure 5.1: Overview of Migration Timings and State Transfer

5.1.1 Mechanics of Migration

Figure 5.1 illustrates aspects of VM migration including execution, state transfer, and performance. Migration is initiated on the source, on which the VM originally executes. The VM is suspended at one point, and then resumed on the destination. State transfer during the course of migration is categorized into two types: pre-copy and post-copy. Pre-copy and post-copy are phases performed before and after the VM resumes on the destination, respectively. Common implementations of live migration, as will be described in detail, are based purely on pre-copy. Associated with the duration of these state transfer modes, there are three key time metrics we define as follows:

- **Down time**: time between the suspension and resume of the VM, during which its execution is stopped.
- **Execution transfer time**: time since the start of migration until the VM resumes on the destination.

- **Total duration**: time since the start and until the end of migration.

In our target contexts, the main objective is to achieve minimal execution transfer time without significant down time. Until execution transfer completes, contending VMs on the source continue to experience degraded performance. Thus, the faster execution transfer is, the more effective the migration operation is in salvaging the performance of both the migrated and other VMs. Sustaining reasonable down time also is important for mitigating the service disruption of the migrated VM. The VM execution halts during this time, rather than continue with performance degradation; extended down time, as a result, can cause network- or application-level time-outs. Finally, the hypervisor on the source needs to maintain the migrated VM's state until the end of total duration. Shorter total duration, therefore, means that allocated resources such as guest memory can be freed and made available to other VMs sooner.

Pre-copy and post-copy phases have their own sources of performance cost. Pre-copy, when overlapped with VM execution, requires tracking state changes to synchronize the destination hypervisor with the latest VM state, a computational overhead known as migration noise [70]. Post-copy, on the other hand, can stall VM execution when the running guest accesses memory contents that have not arrived on the destination. The rationale behind pre-copy and post-copy necessitate these costs. Pre-copy ensures performance upon VM resume at the cost of completing state transfer beforehand. Post-copy, on the other hand, allows fast execution transfer and computational resource freeing on the source, while relinquishing the optimal performance right after VM resume.

5.1.2 Analysis of Live Migration

Live migration is the current standard of migration methods widely adopted by common hypervisors [40, 90]. Its algorithm works as shown in Figure 5.2. Upon initiation, live migration starts sending memory page contents while continuing the VM execution and keeping track of memory content changes. It then proceeds to iteratively retransmit the pages whose content has been dirtied since its last transfer. The purpose of the iteration phase is to minimize down time, thereby optimizing for the liveness of the VM being migrated. While iterating, the algorithm uses the current rate of state transfer to estimate down time, during which the last round of retransmission is performed. If the expected down time is short enough (e.g., 300 ms in qemu-kvm), the iteration phase completes and the VM is resumed on the destination. Implementations can also have additional conditions for preventing migration from taking an excess amount of time. Common examples include a limit on the number of iterations, and high expected down time that steadily ex-

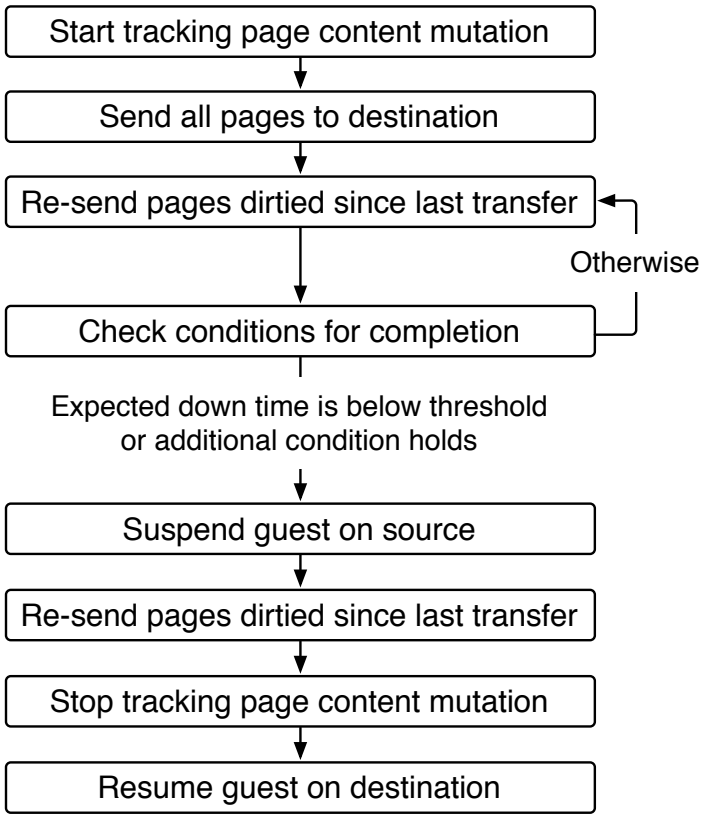
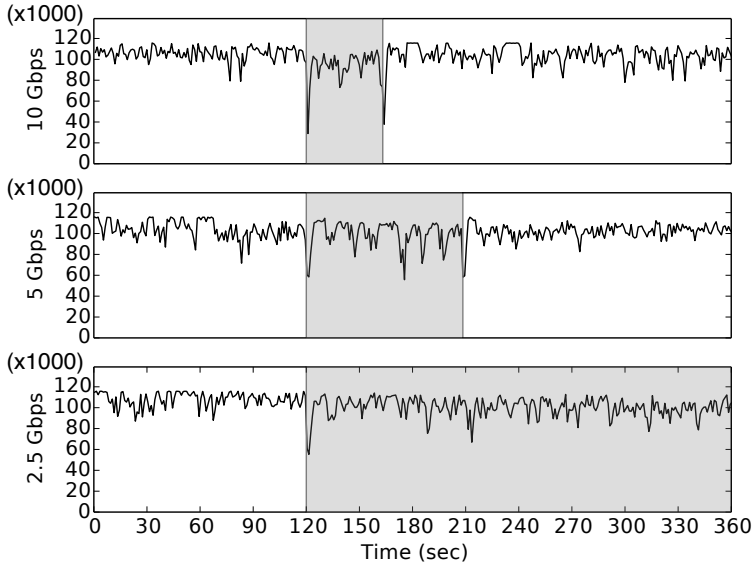


Figure 5.2: Live Migration Algorithm

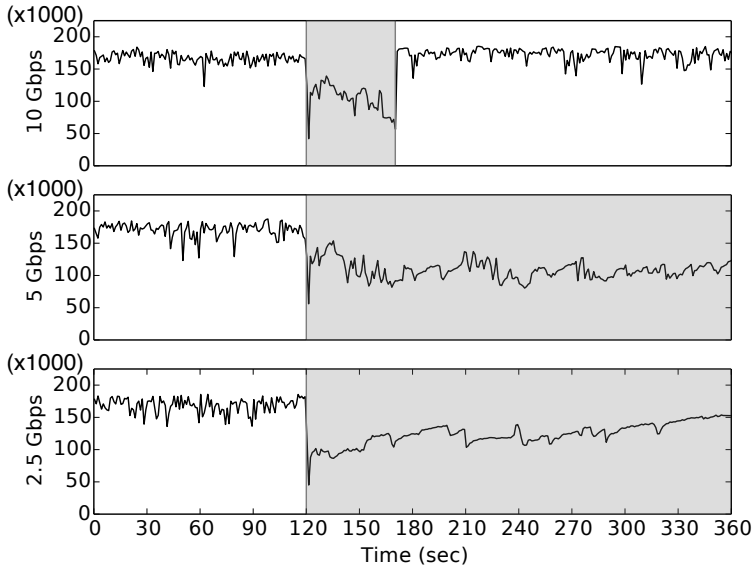
ceeds a threshold. Regardless of the exact form in which these conditions are expressed, common to these parameters of live migration is that they aim to control the maximum duration of the iteration phase. Note that Figure 5.2 illustrates the migration of memory state, assuming the availability of disk state through shared storage. In the rest of this chapter, we also focus on such cases.

Impact of Workloads

Being pre-copy and optimized for down time, live migration handles state transfer while dealing with the guest state changes. Consequently, its behavior depends on the guest workload and the bandwidth available for migration traffic. Figure 5.3 shows the throughput of Memcached server, hosted in a VM, during live migration by qemu-kvm. The



(a) Set-Get Ratio: 1:9



(b) Set-Get Ratio: 5:5

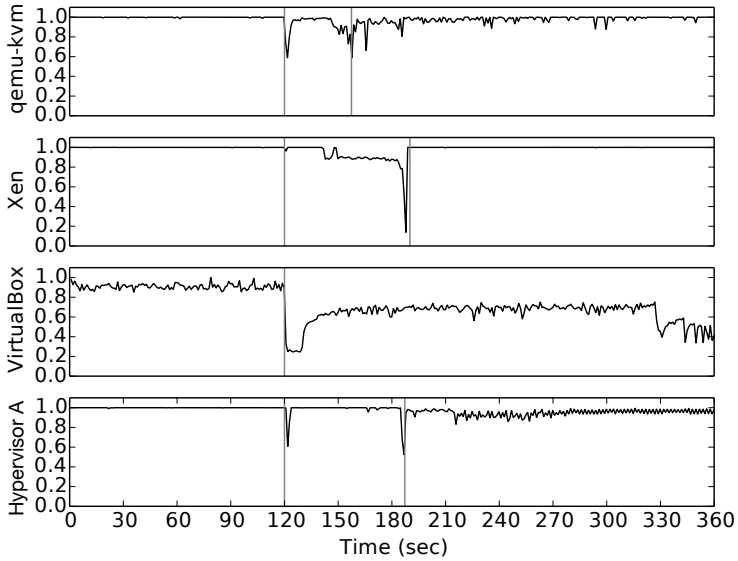
Figure 5.3: Behavior of Migrating Memcached VM with qemu-kvm. The y-axis indicates throughput in operations per second, and the shaded areas represent the duration of migration.

memslap benchmark generates a load for the server, and its set-get ratio and the bandwidth available for migration traffic are varied. Set-get ratio is the ratio of the number of set operations, which specify a value for a given key, to that of get operations, which read the value assigned to a certain key. The other configurations for these measurements are the same as those described in Section 5.4, with the guest allocated 30 GB of memory and the server using 24 GB as its cache. Note that qemu-kvm zeroes out guest memory when setting it up, and live migration compresses each page whose bits are all zeros to one byte accompanied by a header; thus, it avoids sending the unused 6 GB in these measurements.

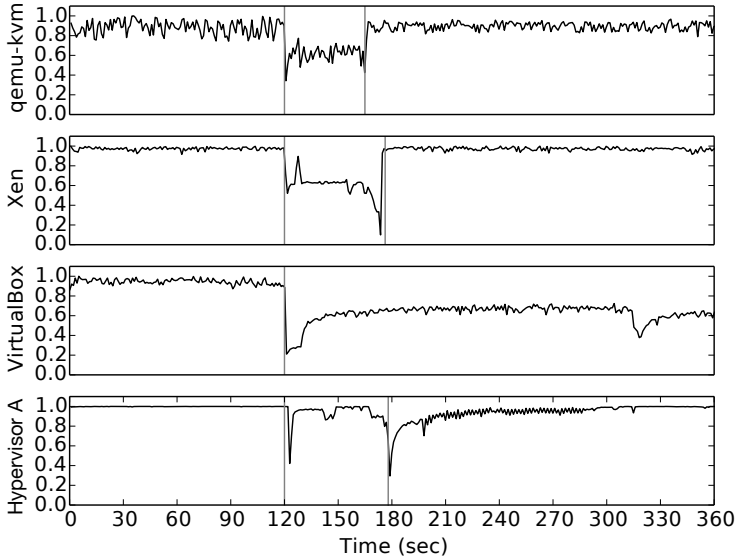
As the available bandwidth for migration traffic decreases, live migration takes more time to complete. This increase is non-linear; with set-get ratio of 1:9, migration finishes in approximately 40 and 90 seconds at 10 and 5 Gbps, respectively. At 2.5 Gbps, it fails to complete in a timely manner. With set-get ratio of 5:5, migration does not complete even at 5 Gbps. This is because expected down time never becomes short enough with the guest workload, and qemu-kvm does not use a hard limit on the number of iterations. In addition, we can observe more throughput degradation during migration with set-get ratio of 5:5 than with 1:9. As the workload generates more memory content changes, dirty state tracking interferes more with it because of trapping memory writes, which are caught more frequently. Finally, even when live migration performs fairly well with set-get ratio of 1:9 and at 10 Gbps, it takes considerably longer than transferring 24 GB over that bandwidth (which takes less than 20 seconds). qemu-kvm's migration functionality is single-threaded, and it takes many enough CPU cycles to transmit state at gigabytes speed while tracking dirty memory pages. Migration can thus easily be a CPU-bound procedure unless special care is taken, for example by parallelization of the code [110].

Commonality among Implementations

The characteristics of live migration explained above are inherent to its algorithm, and therefore to major implementations. Figure 5.4 shows the behavior of migrating a Memcached VM with common hypervisors, qemu-kvm 2.3.0, Xen 4.1.6, VirtualBox 4.3.16 [13], and another major hypervisor (labeled "Hypervisor A"). In the figure, the y-axis represents the throughput of Memcached normalized against the maximum in the corresponding measurement. The VM memory size and server cache size are the same as explained previously: 30 GB and 24 GB. The VM is assigned 4 cores, and migrated over a 10 Gbps link. Note that we used machines different from those for the rest of our measurements, due to hardware accessibility reasons. They were equipped with an Intel Core i7-3770 CPU at 3.4 GHz and 32 GB of memory, running Ubuntu 12.04 with Linux kernel version 3.5.0.



(a) Set-Get Ratio: 1:9



(b) Set-Get Ratio: 5:5

Figure 5.4: Behavior of Migrating Memcached VM with Major Hypervisors at 10 Gbps. The y-axis indicates throughput normalized against the maximum in each measurement, and the shaded areas represent the duration of migration.

As each implementation differs from one another, the performance cannot be directly compared between the hypervisors. In particular, the duration of the iteration phase is determined by parameters, and additional factors such as page content compression also lead to varying performance. For example, VirtualBox does not complete migration in either of the 1:9 and 5:5 set-get ratio cases. Also, unlike qemu-kvm, Xen finishes sooner with set-get ratio of 5:5 than with 1:9, one reason of which is more VM slow-down caused by the migration process itself. The key point of these results, however, is not the absolute performance differences, but the common behavior that the VM lingers on the source for tens of seconds or longer. This total duration exemplifies the cost paid in an effort to minimize down time, and indicates a fundamental trade-off made by pre-copy live migration.

5.1.3 Goals: Metrics of Importance

Live migration thus exhibits undesirable behavior when migrating VMs to resolve their interference and recover their optimal performance. Being a blackbox approach and using pre-copy, it opportunistically waits for a timing at which the VM becomes idle enough to shorten the expected down time. This attempt to minimize the impact on the migrated VM leads to an extended time of VM contention. Driven by this observation, we take a whitebox approach with active guest-host cooperation that 1) frees resources on the source rapidly through fast execution transfer and short total duration, 2) successfully handles loaded VMs without relying on the reduction in their workloads, and 3) salvages the aggregate performance of the VMs under contention, rather than that of the migrated VM only. To summarize, these objectives lead to the following metrics of primary importance:

Execution transfer time:	The metric that decides how fast VM contention can be resolved.
Application service quality:	The impact of migration on the target VM, which encompasses not only its down time but also the disruption of the applications inside.
Total duration:	The time that affects when VM state can be released on the source.

Migration designed for the above metrics is especially desirable in the following cases: (a) optimizing the throughput of VMs running batch-processing workloads, (b) saving the performance of a VM whose service is sensitive to response time, and (c) isolating a malfunctioning VM for diagnosis. Figure 5.5 depicts these three scenarios. The top half of each case illustrates a situation under contention and before migration, and the bottom half

after migration. In case (a), with a group of batch-processing VMs, their total throughput is lowered under contention. Prioritizing it over down time, migration optimizes for the total amount of work despite some disruption of the migrated VM. A good example of workloads that fit this category is big data analytics, ranging from scientific computing to web and social network crawling. In cases (b) and (c), migration is used primarily on behalf of the co-located VMs, rather than the target VM. Case (b) represents a situation in which the VMs run services of different nature: one for which response time is crucial, such as a web server, and the other running a batch-processing job. Resolving their interference fast rescues the former in a timely manner, while improving the performance of the latter from a longer-time perspective. Case (c) involves VMs under circumstances that differentiate the importance of their performance: one under normal operation and the other malfunctioning due to, for example, a DoS attack. Through migration, the malfunctioning VM is rapidly prevented from affecting the other VM, and isolated for diagnosis of the problem. This case has the most emphasis on the urgent salvation of the non-migrated VM instead of the migrated VM. In all these cases, migration with appropriate characteristics would allow the VMs to be efficiently co-located, and to be allocated new resources when experiencing high loads.

Also, we note that it is not our goal to outperform live migration in every aspect, such as sustaining the availability and accessibility of the migrated VM and migrating idle or lightly loaded VMs. It is more reasonable to employ live migration in these cases, which it already addresses well.

5.1.4 Current Execution Knowledge

To address the challenges of urgent VM migration under contention, we exploit current execution knowledge about the VM. Its use is motivated by a few properties of the target contexts of this problem. First, unlike the delivery of virtual appliances, we work with running VMs that have a constantly demanding workload. In such situations, the characteristics of resource usage by the guest changes over time. Achieving the efficiency of the migration operation, therefore, requires that it reflect the latest state of the VM, rather than a single point in the past. Second, we expect that the migration for resolving contention is a reactive operation that cannot be planned in advance. The load balancing decision should be feasible dynamically, and honored within an adequately short time period. This requirement necessitates the use of execution knowledge that is accurate and applicable within this time frame. Namely, the execution knowledge must be gathered and used at the time of migration.

This rationale also justifies the choice of the guest OS as the source of execution knowl-

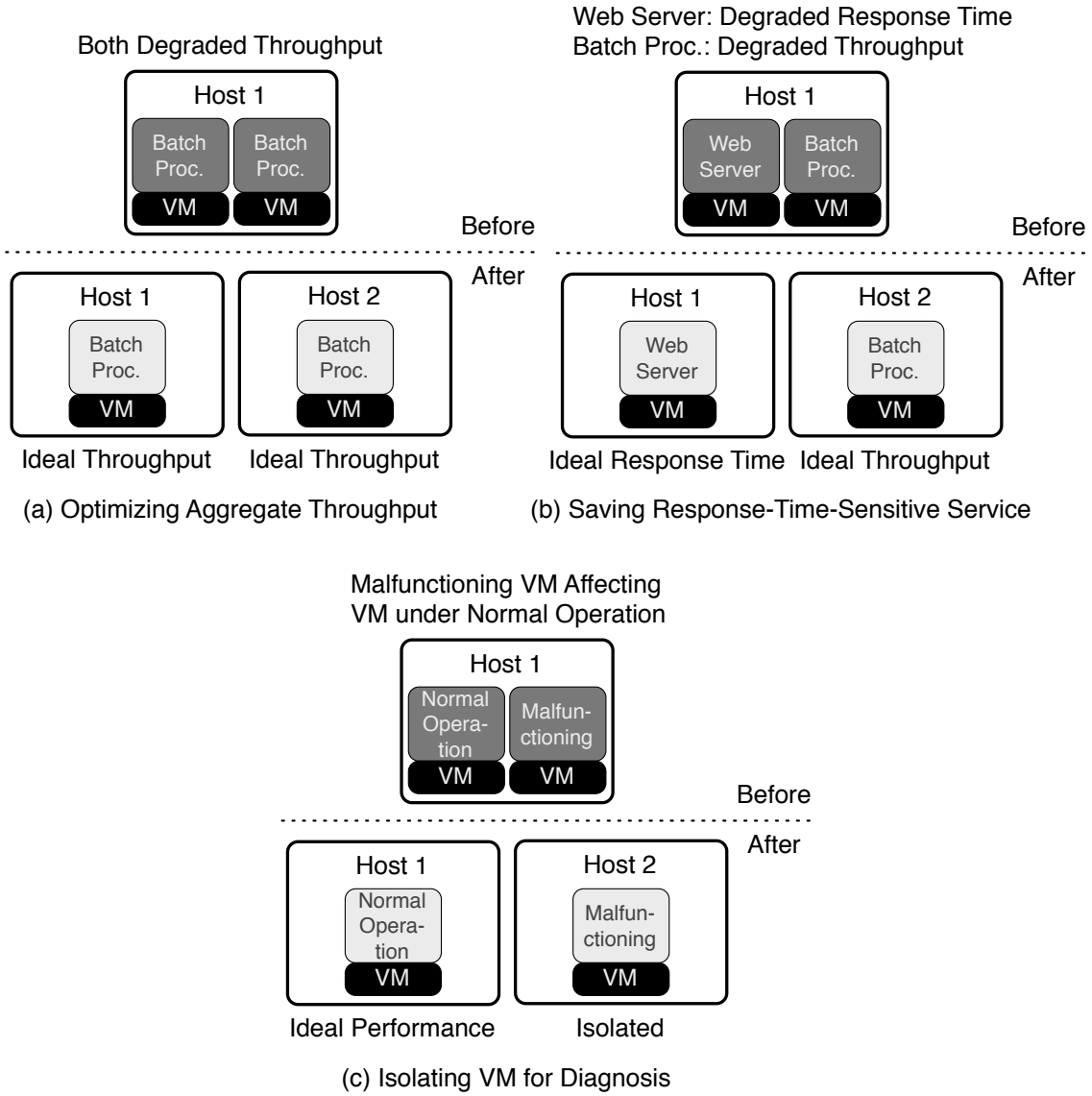


Figure 5.5: Scenarios Demanding Migration with Fast Execution Transfer. In each case, the top half and bottom half illustrate VMs before and after migration, respectively.

edge. The guest OS manages the memory allocated by the hypervisor for the VM, and therefore knows the meanings and importance of the pages in the allocated memory. Approaches at the hypervisor level are also possible, and expected to be less intrusive to the systems software hierarchy in virtualized environments. However, such approaches do not have access to the system information available inside the guest OS. As a result, they require monitoring or inference of the nature of the guest workload to make informed decisions that lead to efficiency in migration. As in the case of live migration, insights obtained in this indirect way typically come at the cost of a non-trivial time spent collecting them; it is undesirable to sacrifice the timeliness of gathering the necessary information. We will show that our direct approach at the guest OS level, instead, is capable of passing execution knowledge to the hypervisor in a timely fashion.

5.2 Enlightened Post-Copy Migration

Our approach to the above goals, called enlightened post-copy, exploits guest cooperation and post-copy-style state transfer. We derive the key ideas behind this approach specifically from our goals. First, minimizing execution transfer time requires that VM execution be immediately suspended on the source and resumed on the destination. This early suspension upon the start of migration also ensures minimal total duration, because the frozen VM state necessitates no retransmission as done in live migration. Therefore, post-copy follows naturally as the desirable method of state transfer. Second, fast performance recovery of the migrated VM requires that the part of its state needed for its current workload arrive at the destination as early as possible. The guest's enlightenment is the key that enables identifying this part of the VM state; with state transfer following the instructed prioritization, the VM on the destination can start recovering its performance without the completion of entire state transfer, and thus before the total duration of migration.

Figure 5.6 illustrates the workflow of enlightened post-copy. When migration is initiated, the hypervisor makes a request for enlightenment to the guest OS. The guest OS traverses its data structures and prepares priority information of the memory pages. Once the priority information is available, the guest OS notifies the hypervisor. The hypervisor then suspends the VM on the source, and resumes it on the destination immediately after sending the device state necessary for the resume operation. As the VM starts execution, the hypervisor parses the priority information and accordingly transfers the remaining memory page contents to the destination; it attempts to proactively push as many pages as possible before the access to them.

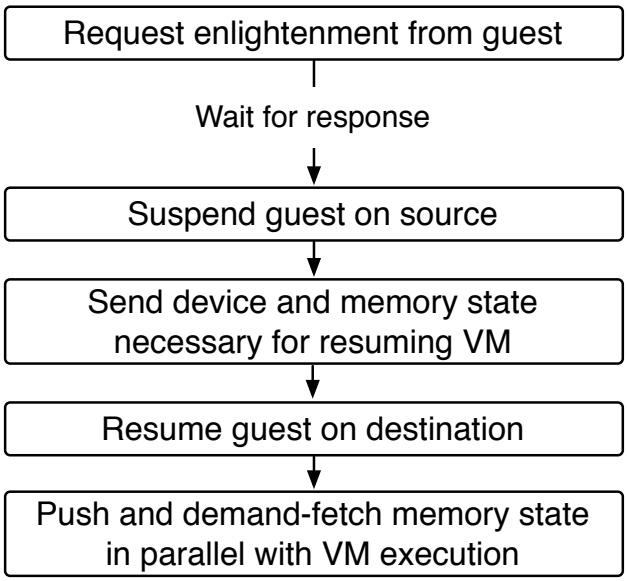


Figure 5.6: Workflow of Enlightened Post-Copy Migration

5.2.1 Guest's Enlightenment

In enlightened post-copy, the guest identifies those memory pages containing the working set of the currently active processes. As a design principle, the guest OS should be able to obtain a list of these pages without incurring noticeable overhead. Otherwise, the approach does not justify the guest instrumentation due to the resulting performance loss. As the types of memory page classification, therefore, we use straightforward notions such as the code and data of the OS kernel and running processes. Such bookkeeping information of memory pages is already available in the OS for its regular tasks, and re-usable without significant implementation effort for the purpose of migration.

For prioritized state transfer, the general idea is to transfer memory pages essential for running the guest system itself, those for the actively running processes, and then the other less critical pages such as the kernel page cache and those for the non-active processes. Also, we can eliminate the transfer of the memory pages that are not allocated by the guest OS for any use, because the actual contents of such pages do not affect the correctness of guest execution.

The guest needs to prepare these types of information, as enlightenment to the hypervisor, in two distinct forms. The memory page priorities can be determined by a one-time

operation upon the request by the hypervisor. There is no need for always tracking them during the course of the guest's normal operation. On the other hand, keeping track of allocated and non-allocated memory pages requires real-time processing, performed with or without migration, that maintains the information in a manner easily passed to the hypervisor. The reason is that the source hypervisor needs to know the exact allocation by the guest OS right at the time of VM suspension, for the destination hypervisor to construct a consistent memory image. For the one-time operation, the associated costs are that of guest-host communication delay upon migration start, and the impact of the in-guest processing on performance. For the real-time processing, the cost is the overhead added to relevant memory management operations of the guest OS. Minimizing these costs motivates the use of the above types of enlightenment, which are adequately informative but not excessively fine-grained.

5.2.2 Integration into State Transfer

The source hypervisor can integrate enlightenment into state transfer in a straightforward manner, because of the use of post-copy. Since the VM state is frozen at the start of migration, enlightenment at that time reflects its latest state before execution transfer, from which the VM resumes on the destination. After receiving enlightenment and suspending the VM, the source hypervisor pushes the memory pages as instructed by the guest OS. While the destination hypervisor receives the memory pages, it also issues demand-fetch requests to the source for those that are accessed by the guest before their arrival. Although their arrival may incur delays due to the interference with the push traffic, these demand fetches help reduce VM execution stalls due to the divergence of the push order from the actual access order by the guest.

5.2.3 Design Trade-Offs

The design of enlightened post-copy is in good contrast to that of common live migration based on pre-copy. Enlightened post-copy targets VMs constantly under loads, while live migration expects idleness from them. Enlightened post-copy, therefore, employs a state transfer method that enables timely load balancing through fast physical resource reallocation. Down time and execution transfer time are expected to be rapid, and total duration corresponds to the one-time transfer of the entire state. At the same time, the disruption of the migrated VM's performance spans a longer period than down time itself, since post-copy is used. Guest cooperation is the key to alleviating this disruption.

On the other hand, live migration focuses on one aspect of the migrated VM's performance, down time. Being a guest-agnostic approach without an external source of knowledge, it relies on letting the VM stay on the source and tracking dirtied memory pages. Execution transfer time is equivalent to total duration; these time frames become longer when more iterations are done. The sole use of pre-copy ensures the migrated VM's performance on the destination, since all the state resides there on VM resume. Thus, down time approximately represents the duration of application-level disruption. However, dirty page tracking incurs a certain cost while the VM lingers on the source. Results in Section 5.4 demonstrate the effects of these trade-offs made by enlightened post-copy and live migration.

5.3 Architecture and Implementation

We implemented enlightened post-copy on guest Linux 3.2.0 and hypervisor qemu-kvm 2.3.0. Figure 5.7 shows its architecture. When the source hypervisor initiates migration, it sends a request to guest Linux through a custom VirtIO device [21] (Step 1). The guest OS driver for this virtual device triggers enlightenment preparation, which scans data structures (Step 2) and writes priority information in the priority bitmap (Step 3). Page allocation information is always kept up-to-date in the free bitmap, so that its content is valid whenever the the hypervisor suspends the VM. These bitmaps maintained in the guest's memory facilitate the processing by the hypervisor; they are an abstract enough representation of the passed information, and the guest OS can avoid communicating it through the VirtIO device and instead have the hypervisor directly parse it. When the priority bitmap has been written, the guest OS notifies the hypervisor through the VirtIO device (Step 4). The hypervisor then sends to the destination the device state, including some in the guest memory, which is used by the destination hypervisor for the initial VM set-up. Finally, it starts transferring the remaining page contents in the prioritized order (Steps 5 and 6). On the destination, the hypervisor resumes the VM once the device state has arrived. While receiving the pushed page contents, it writes them into the guest memory. When the guest OS accesses pages whose content has not yet been received, it generates demand-fetch requests to the source hypervisor. On the source, the hypervisor frees all the VM resources once all the page contents have been sent.

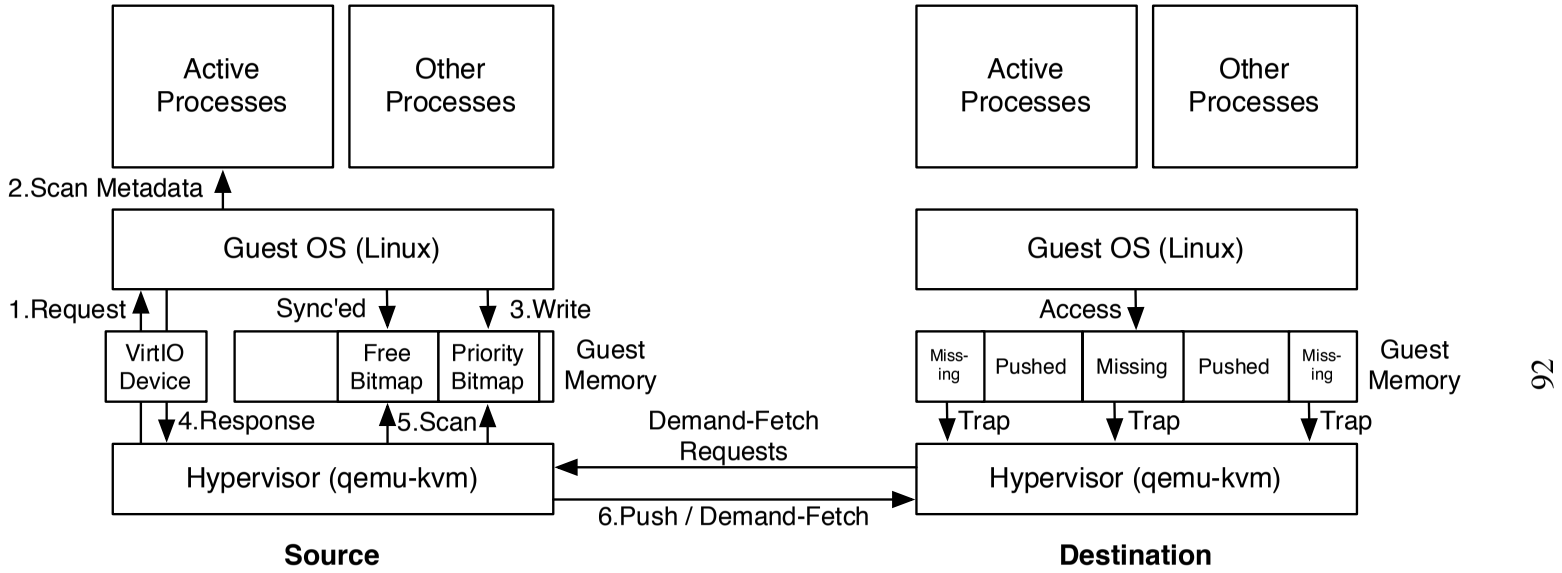


Figure 5.7: Implementation of Enlightened Post-Copy. The numbers represent the steps in order during the course of migration.

5.3.1 Guest OS

In our guest Linux, memory management and process scheduling code is instrumented to label each memory page with a priority level. The instrumentation follows naturally in the relevant existing parts of the source code, and requires only a modest number of changes to the original kernel.

Enlightenment Preparation

Taking advantage of memory management information that already exists, the guest OS classifies the memory pages in use into the following priority categories:

- Kernel: kernel executable code
- Kernel_Allocated: allocated for kernel use
- Memory_I/O: used for memory-mapped I/O
- Page_Table: page table of active process
- User_Code: executable page of active process
- User_Data: non-executable page of active process
- File_Active: active cache of file
- File_Inactive: inactive cache of file
- Other: not belonging to any of the above

The system-wide categories, such as Kernel, Kernel_Allocated, Memory_I/O, File_Active, and File_Inactive, are derived from kernel data structures or through the flags of page frame descriptors (e.g., `struct zone` and `struct page`). For the process-specific categories, the enlightenment preparation code parses the virtual memory area descriptors of each active process (`struct vm_area_struct`). These categories are each assigned a numerical value, in a descending order of priority from the top to the bottom in the above list. This order is decided such that the core system service, the active processes, and caching by the OS are given priority in that order. If a particular page belongs to multiple categories, it is treated with the highest of these priorities. Pages such as those belonging to the inactive processes and those used for the priority bitmap itself belong to the Other

category. The bitmap simply contains the priority values, without the hypervisor needing to understand their exact semantics.

In order to decide the group of active processes, the scheduler maintains the last time each process was scheduled for execution (in `struct task_struct`). A process is considered active if it has been run recently at the time of generating enlightenment. In our implementation, we empirically use a threshold of the past 16 seconds for this purpose, considering the order of seconds migration is roughly expected to take.

When the guest OS receives a request from the hypervisor, the guest OS runs code that generates the priority bitmap as explained above. Once it notifies the hypervisor, it is ensured that the guest can be suspended with the bitmaps available for parsing in its memory. Note that the free bitmap is always kept up-to-date and ready for processing. The response from the guest OS includes the addresses of the two bitmaps, and the hypervisor scans these locations while performing state transfer.

Implementation Cost

The modifications to the core Linux kernel code are minimal, mostly under `mm/` and `kernel/` in the source tree. Maintaining the free bitmap requires adding at most several lines of code in 16 locations. Keeping track of the last schedule time for each process requires a few variable assignments added in the scheduler code. Marking each page with kernel or user allocation needs less than 15 lines of code. These operations only add atomic variable assignments in the code paths of memory allocation and process scheduling. As shown in our experiments, compared to the original kernel, these minor modifications incur negligible performance overhead. The VirtIO device driver and the priority bitmap preparation code, which make use of the instrumentation, are implemented as loadable kernel modules.

5.3.2 Hypervisor

On the source, the hypervisor scans the bitmaps provided by the guest. It first scans the free bitmap and sends the corresponding page frame numbers in a packed format, along with the device and some memory state, right after which the VM is resumed on the destination. Next, the hypervisor traverses the priority bitmap and starts pushing the page contents over a TCP connection. The transfer is performed in rounds, starting with the Kernel pages and ending with the Other pages. While this push continues, a separate thread services demand-fetch requests from the destination hypervisor over another TCP

connection. Note that while we attempt to give a higher priority to the demand-fetched pages through the TCP_NODELAY socket option, the push transfer can still interfere with their arrival timings.

On the destination, the hypervisor registers userfault [20] handlers with the guest memory region. Userfault is a mechanism on Linux that enables a user process to provide a page fault handler of its own for specified pages. This interposition enables post-copy state transfer. The userfault handler is first registered for the entire main memory of the VM when the hypervisor starts on the destination. On the receipt of the unallocated page information, the handler is removed from the respective addresses. Then, as the memory page contents arrive, they are written to the corresponding addresses and the userfault handler is unregistered from these addresses. When the guest accesses these memory pages whose content is already available, no further interposition by the hypervisor is carried out. On access to a page whose content is still missing, the hypervisor issues a demand-fetch request to the source hypervisor.

5.4 Evaluation

We performed experiments to demonstrate the effectiveness of enlightened post-copy in resolving resource contention and the performance trade-offs resulting from its design principles. Our experiments address the following points. First, we show how enlightened post-copy salvages the throughput of contending VMs through end-to-end results, while comparing them to those of original live migration of qemu-kvm 2.3.0. Next, we investigate the efficacy of our approach in robustly dealing with workloads by examining the performance of two baseline methods, stop-and-copy and simple post-copy, as reference points. Finally, we show the cost of enlightened post-copy in terms of state transfer amounts and guest OS overhead.

5.4.1 Set-Up and Workloads

Figure 5.8 shows our experiment set-up. VMs are migrated between a pair of source and destination hosts, which are connected through a backend 10 Gbps network for migration traffic. They are also connected on this network to an NFS server that stores VM disk images. The client machines use a separate 1 Gbps network for user traffic to and from the VMs. In the experiments except those with idle VMs, initially two VMs are running and contending with each other on the source machine. We migrate one of the VMs to the destination machine, after which each VM has its own dedicated machine. The VM hosts

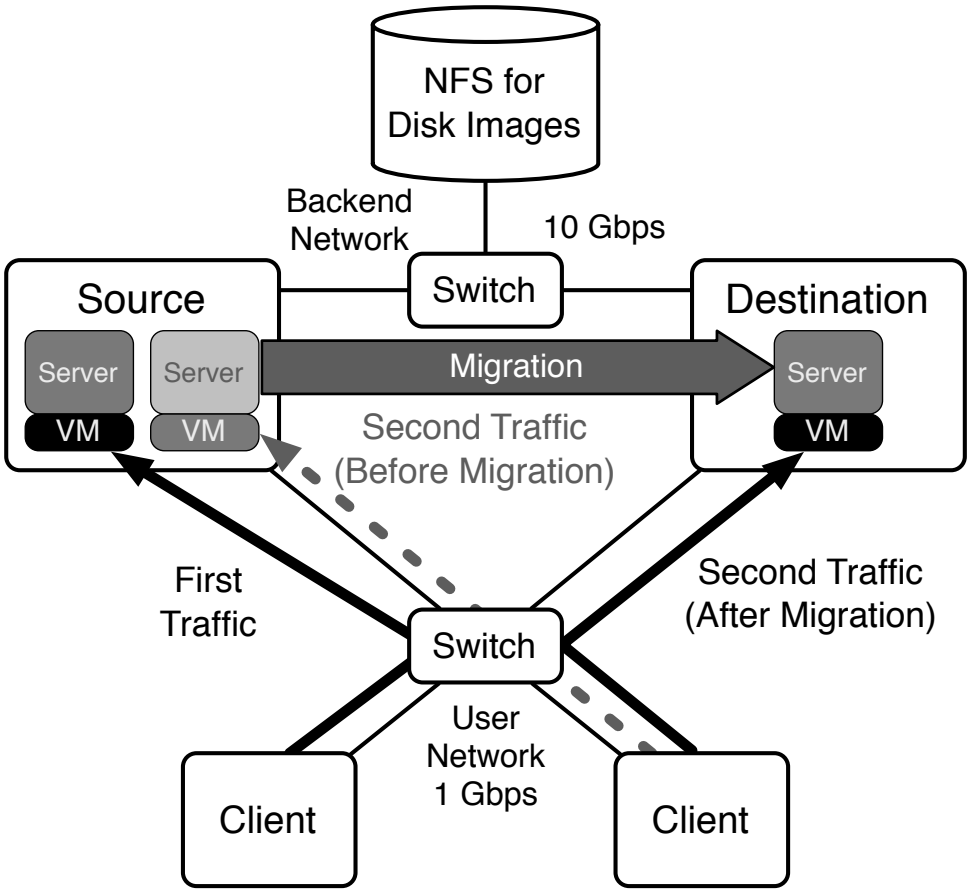


Figure 5.8: Experiment Set-Up

and client machines are each equipped with two Intel Xeon E5-2430 CPUs at 2.20 GHz and 96 GB memory, running Ubuntu 14.04. The VM hosts execute Linux kernel 4.1.0-rc3 with a userfault patch applied, while the other machines run version 3.16.0. As userfault currently does not support the use of huge pages, they are disabled in our measurements. Time on the machines is synchronized via an NTP server, and the backend bandwidth between the VM hosts is controlled using Linux Traffic Control for measurements at 5 Gbps and 2.5 Gbps. The VMs run Ubuntu Server 12.04 with unmodified kernel 3.2.0 in all the cases except those with enlightened post-copy, in which we use our modified version of the kernel.

We use the following workloads that exhibit different types of resource intensity and

reveal performance trade-offs made by enlightened post-copy:

Memcached: The VMs run an in-memory key-value store, Memcached 1.4.13 [8], and the clients execute its bundled benchmark memslap 1.0, which is modified to report percentile latencies. The VMs are each allocated 30 GB of memory and 8 cores, with Memcached configured with 4 threads (due to its known scalability limitation) and 24 GB cache. We first run the benchmark against Memcached to fill up its cache, and then perform measurements with concurrency level of 96 and set-get ratio of 1:9. At the time of migration, approximately 24 GB of memory is in use, almost all of which is by Memcached.

MySQL: The VMs run MySQL 5.6, and the clients execute OLTPBenchmark [12] using the Twitter workload with scale factor of 960. The VMs are each allocated 16 cores and 30 GB of memory, and MySQL is configured with a 16 GB buffer pool in memory. The concurrency of OLTPBenchmark is set to 64. After generating the database contents, we execute Tweet insertions for 25 minutes and then the default operation mix for 5 minutes as a warm-up. At the time of migration, MySQL uses approximately 17 GB of memory, and almost all of the 30 GB memory is allocated by the guest OS for use.

Cassandra: The VMs run a NoSQL database, Apache Cassandra 2.1.3 [19], and the clients use YCSB [22] 0.1.4 with 24 threads and core benchmark F, which consists of 50% read and 50% read-modify-write operations. The VMs are each configured with 16 cores and 30 GB of memory. Before measurements, the benchmark is run for approximately 10 minutes to warm the servers up. At the time of migration, the server uses around 8.4 GB of memory out of 12 GB in use by the guest OS.

In the above workload configurations, Memcached is the most memory- and network-intensive, while consuming relatively low CPU resources. Also, the VM's memory is almost exclusively used by the server process itself. MySQL is more CPU-intensive, and also less memory-intensive in terms of the access footprint per unit time. Finally, Cassandra is the most compute-intensive among these workloads, making CPUs the source of contention. In the MySQL and Cassandra cases, the guest OS uses a non-trivial amount of memory in addition to that allocated by the server processes themselves. These characteristics make Memcached the worst case, and MySQL and Cassandra more winning cases for enlightened post-copy in comparison to live migration.

5.4.2 End-to-End Performance

In this section, we compare application-level performance of the three workloads during migration with enlightened-copy and live migration. In addition to the throughput of the server applications, we also report the impact of migration on application-level latency.

Memcached

Figure 5.9 compares Memcached throughput of enlightened post-copy (labeled "EPC") and live migration (labeled "Live") under varied bandwidth. The y-axis shows operations per second in thousands (x1000), and total duration of migration is shown as shaded areas. The dark lines indicate the performance of the migrated VM, and gray lines are that of the contending VM. The source of contention is user traffic handling by the source hypervisor. As Memached accounts for almost all the guest memory pages in use (which is categorized into User_Data) and accesses them at a great speed, it leaves little room for memory prioritization through enlightenment. Thus, the performance recovery during enlightened post-copy migration is not significant because it requires most pages for the Memcached server to be present. Immediate execution transfer, still, lets the contending VM recover its performance as soon as the migration starts. On the other hand, live migration handles the read-mostly workload relatively well. At 10 Gbps, it finishes almost as soon as enlightened post-copy does. However, it performs more state retransmission as bandwidth becomes lower, and at 2.5 Gbps it fails to finish while the benchmark continues. Note that, because of the migration thread causing the saturation of a core at 10 Gbps, the results at this speed are not as good as can be expected from the 5 Gbps results.

The latency characteristics of the 10 Gbps measurements are shown in Figure 5.10. The top two graphs present the 90th percentile latency of the server responses over time. The latency stays roughly between 1 and 2 ms before migration, and around 1 ms once it completes. Live migration sustains mostly good latency until the end of migration. Enlightened post-copy leaves the 90th percentile latency close to 1 second right after the start of migration, while it starts to decrease as more state arrives at the destination. The bottom two graphs show CDFs of the response times during the same 5-minute period. Despite its longer tail to the right side, enlightened post-copy still maintains the curve of the migrated VM close to that of the contending VM and those with live migration. While the differences in throughput should also be considered when interpreting these results, this means that the impact on latencies of the served requests is moderate at the 5-minute granularity.

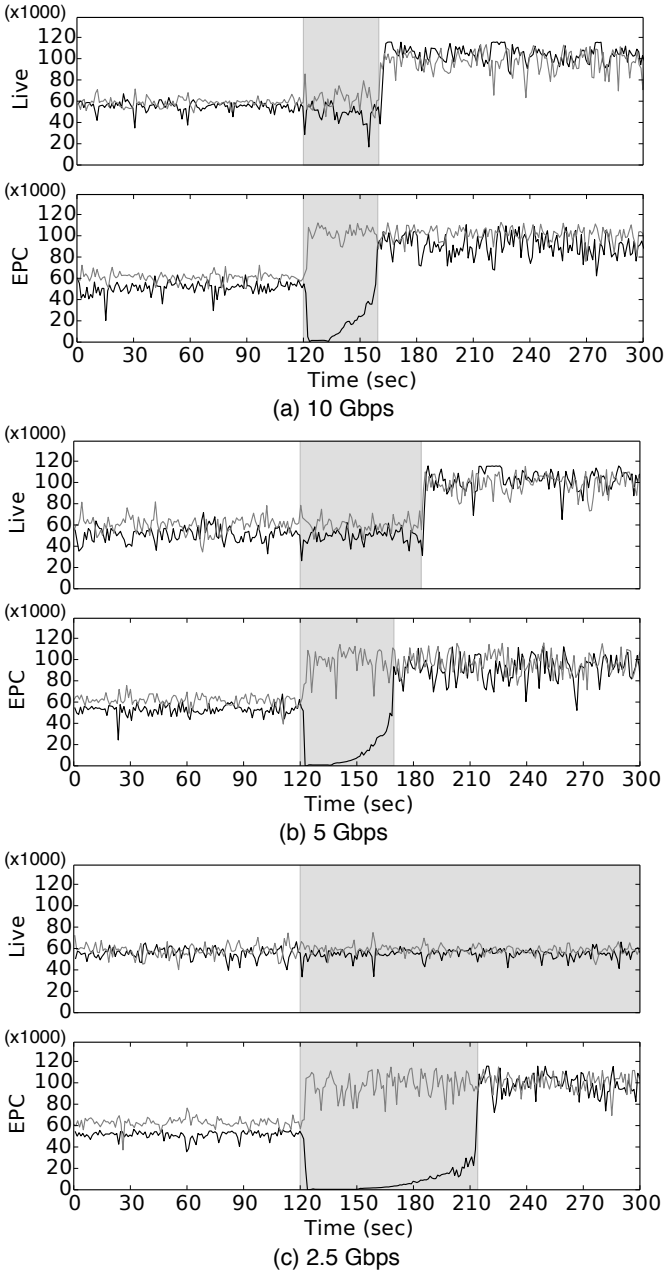


Figure 5.9: End-to-End Results with Memcached. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the shaded areas the duration of migration.

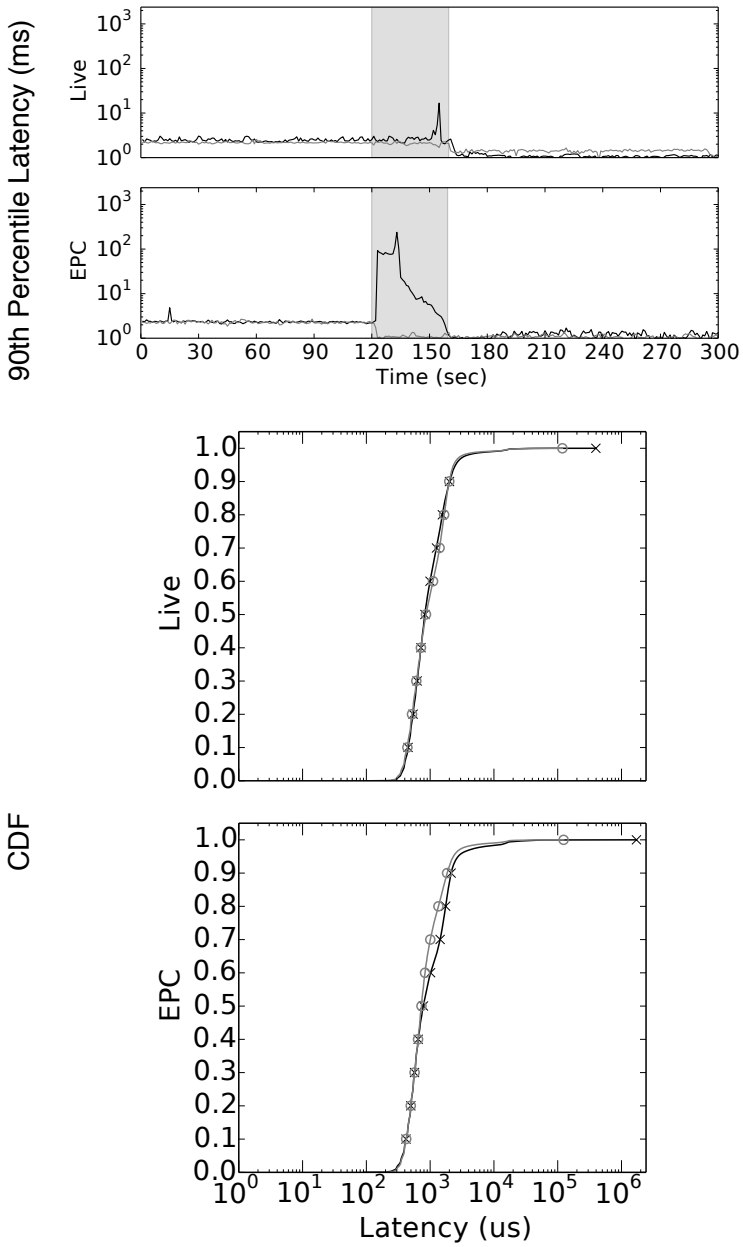


Figure 5.10: Latency with Memcached at 10 Gbps. The black and gray lines correspond to the migrated and contending VMs, respectively. The top two graphs show the 90th percentile latency over time on a log scale, with the shaded areas indicating the duration of migration. The bottom two figures are CDFs of the response latencies during the 5-minute period, with markers at every 10th percentile.

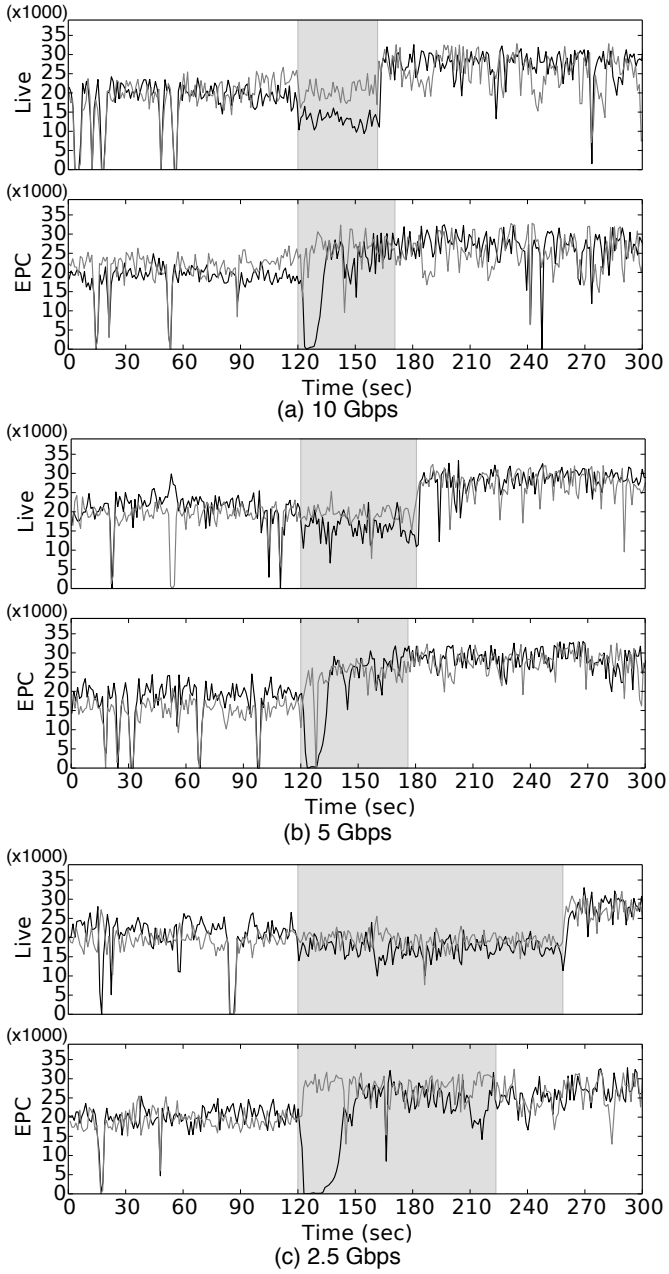


Figure 5.11: End-to-End Results with MySQL. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the shaded areas the duration of migration.

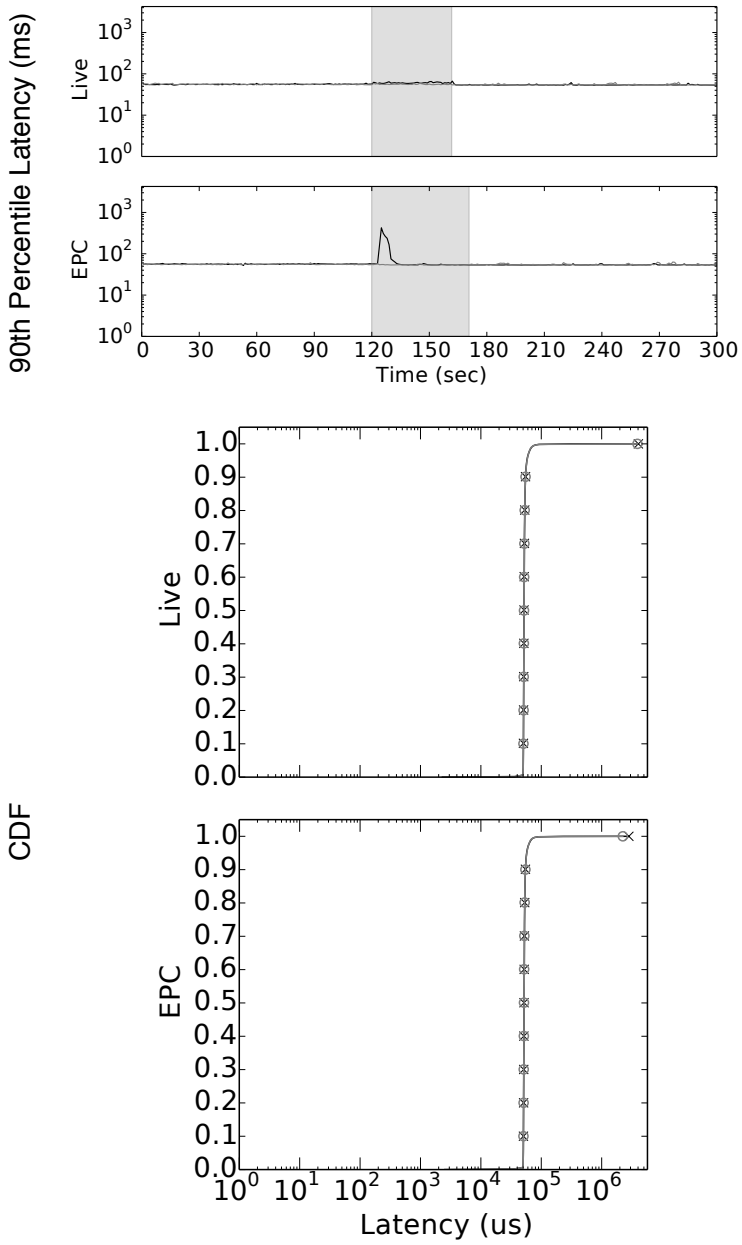


Figure 5.12: Latency with MySQL at 10 Gbps. The black and gray lines correspond to the migrated and contending VMs, respectively. The top two graphs show the 90th percentile latency over time on a log scale, with the shaded areas indicating the duration of migration. The bottom two figures are CDFs of the response latencies during the 5-minute period, with markers at every 10th percentile.

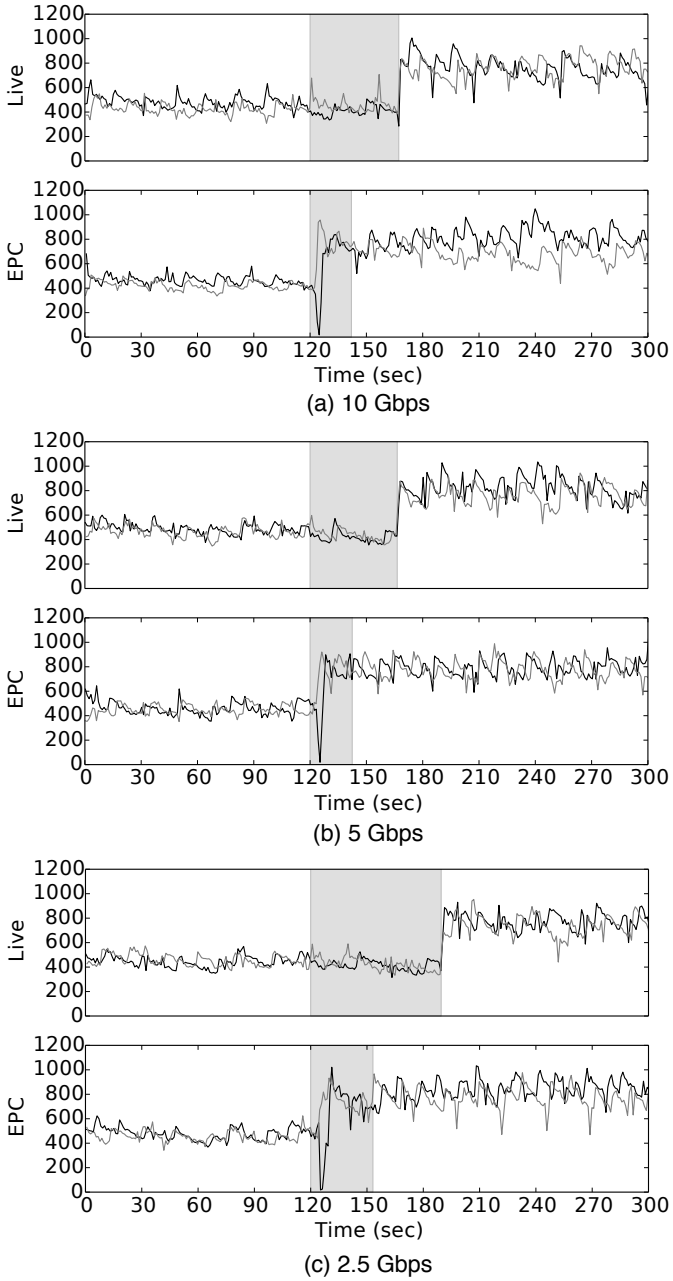


Figure 5.13: End-to-End Results with Cassandra. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the shaded areas the duration of migration.

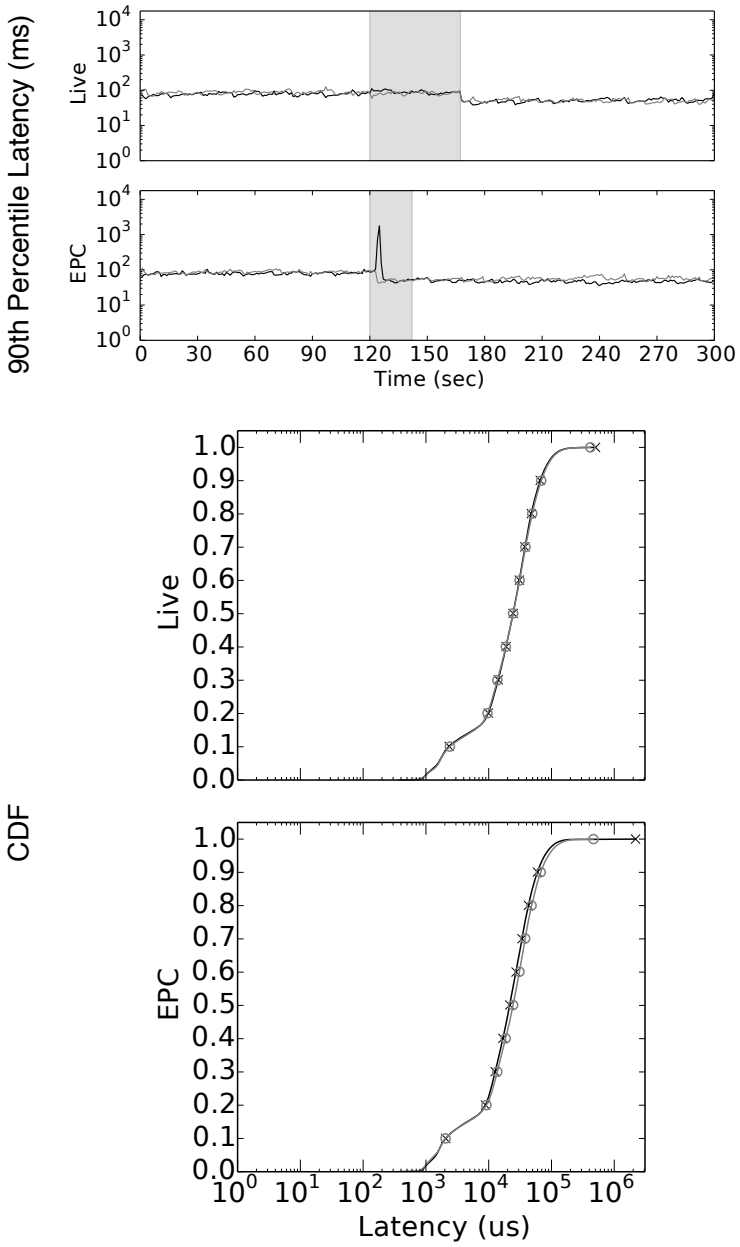


Figure 5.14: Latency with Cassandra at 10 Gbps. The black and gray lines correspond to the migrated and contending VMs, respectively. The top two graphs show the 90th percentile latency over time on a log scale, with the shaded areas indicating the duration of migration. The bottom two figures are CDFs of the response latencies during the 5-minute period, with markers at every 10th percentile.

MySQL

Figure 5.11 shows the throughput results with MySQL. Although the workload is less network-intensive than Memcached, multiplexing user traffic between the VMs on the source causes contention. We also attribute to this bottleneck the ephemeral performance drops that are observed especially before migration. As the workload has more memory access locality, as well as memory allocated besides the cache of MySQL itself, enlightened post-copy gains significantly from prioritized state transfer. The throughput of the migrated VM starts recovering shortly after the migration start, and well before its total duration. In addition to taking longer as the workload size increases with respect to bandwidth (i.e., at lower bandwidth), live migration also exhibits more interference with the throughput of the migrated VM at higher bandwidth. The reason is that the workload is fairly CPU-intensive, and that the migration thread performs dirty state checking more frequently per unit time. Also, unlike all the other cases, live migration completes sooner than enlightened post-copy at 10 Gbps. As the interference of the hypervisor slows the guest, it consumes less computational resources besides those spent for network transfer than enlightened post-copy does. As a result, live migration can better utilize the bandwidth.

The latency results for the workload are shown in Figure 5.12. The 90th percentile latency with enlightened post-copy recovers quickly as the throughput does, lowering to the level of a VM without contention before the completion of migration. The CDFs also indicate that the response time distributions are comparable between the two methods, including the tails to the right representing the maximum latency.

Cassandra

Finally, Figure 5.13 shows the results with Cassandra. This workload makes the CPUs on the source the bottleneck for the VMs before migration. Its total duration not being affected by the resource intensity of the workload, enlightened post-copy finishes as soon as the amount of memory in use has been transferred. It also starts recovering the throughput of the migrated VM halfway through the migration process. With severe CPU contention on the source, live migration is prevented from performing dirty state checking and state transmission frequently. Thus, we do not observe its interference with the migrated VM's throughput, but instead see total duration heavily penalized at higher bandwidth; effective state transfer rate stays low enough that the duration differs only slightly between 10 and 5 Gbps. Overall, the difference in the duration between the two methods is more significant than with the other workloads.

As shown in Figure 5.14, the good performance of enlightened post-copy is also re-

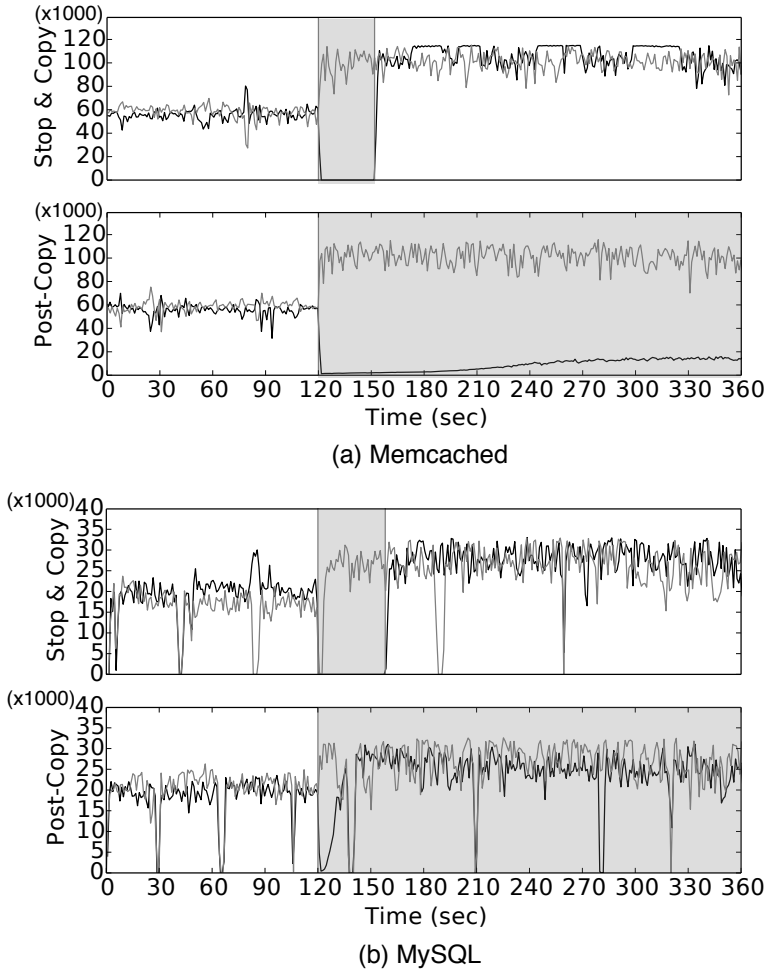


Figure 5.15: Behavior of Baseline Approaches. The y-axis indicates throughput in operations or transactions per second. The black and gray lines represent the migrated and contending VMs, respectively, and the shaded areas the duration of migration.

flected in the latency results. The 90th percentile latency increases for a short period with enlightened post-copy, and soon drops to the ideal level without the contention. Also, the response time distributions of enlightened post-copy and live migration compare well to each other. Except for the right tail of the migrated VM being a little longer with enlightened post-copy, the two methods show similar distribution curves.

5.4.3 Comparison with Baseline Approaches

We have so far compared enlightened post-copy with live migration based on pre-copy, which is predominantly used in today's virtualized environments. We further describe how effectively our design achieves its objectives by comparison to two fundamental approaches: stop-and-copy [103, 117] and simple post-copy. Stop-and-copy is an early form of migration that stops the VM, transfers all its state, and resumes the VM, in a sequential manner. It achieves the shortest total duration possible at the cost of making down time equivalently long. Simple post-copy solely uses demand fetches. It transfers only those memory pages that are being accessed by the guest on the destination, making each access incur an RTT between the hosts. These approaches can be considered as extreme design points: stop-and-copy as live migration that eliminates iterations for the sake of minimal duration, and simple post-copy as enlightened post-copy without, or with completely ineffective, enlightenment. They thus serve as baselines that reveal the benefits of using the sophistication in enlightened post-copy.

Figure 5.15 illustrates the behavior of stop-and-copy and simple post-copy with the Memcached and MySQL workloads at 10 Gbps. The two workloads exemplify cases in which they perform well or poorly compared to enlightened post-copy (whose corresponding cases are shown in Figures 5.9 (a) and 5.11 (a)). Stop-and-copy works relatively well for Memcached, and poorly for MySQL. Its performance is determined by the state transfer amount, regardless of the workload, while enlightened post-copy copes better with the MySQL workload than with the Memcached workload. The gain by enlightened post-copy, therefore, becomes clearer in the MySQL case. Simple post-copy is ineffective for Memcached and fairly adequate for MySQL. It significantly impacts the Memcached performance once the VM starts on the destination, as the cost of page retrieval is prohibitive for the memory-intensive workload. MySQL, on the other hand, exhibits enough memory access locality to prevent this cost from significantly affecting its performance. As a result, enlightened post-copy shows a clear advantage in the Memcached case. In summary, stop-and-copy and simple post-copy have cases they can handle and those they cannot; enlightened post-copy performs comparably to them in their best cases, and outperforms them considerably in the other cases.

5.4.4 Costs of Enlightenment

Enlightened post-copy targets VMs under loads and makes explicit design trade-offs. One question that arises is the overhead incurred due to its design when used in other situations. Table 5.1 shows time and state transfer statics of migrating an idle VM with 30 GB memory over 10 Gbps. The guest OS uses approximately 1 GB of memory, with no user applications actively running. In part (a), the columns from left to right indicate guest communication time for obtaining enlightenment, time until the VM is suspended on the source, execution transfer time, and total duration. Although enlightened post-copy pays the price of communicating with the guest OS, the cost is insignificant in this idle VM case. Live migration, even when the VM is idle, needs to scan the entire guest memory and thus takes some time until completion. Overall, enlightened post-copy is no worse than live migration in terms of the time metrics. Part (b) in the figure shows the amount of state transfer by the transfer method used. "Free memory information" for enlightened post-copy represents how much data was sent to inform the destination hypervisor of all the unallocated memory pages. Since enlightened post-copy performs one-time transfer and live migration needs little retransmission, they transfer similar amounts in total.

In order to measure the real-time cost of tracking page allocation in the free bitmap, we ran a microbenchmark program inside the guest. The program performs repeated memory allocation, as fast as possible, in chunks of 1000 individual 4KB malloc() and free() calls each. With the original Linux kernel and our modified kernel, one pair of these calls took 1.394 us and 1.455 us (4.4% increase), respectively. As demonstrated in the preceding results, this difference typically has a negligible impact on applications because they do not allocate and free memory as frequently as the program does.

Summary of Results

The main properties of enlightened post-copy, as demonstrated by the end-to-end results, are summarized as follows. First, as a result of fast execution transfer, the migrated VM ceases to contend for resources soon after the migration is initiated. The other VMs are then able to use the freed resources. Second, the performance of the migrated VM starts recovering before the completion of migration, especially rapidly with MySQL and Cassandra. Enlightenment achieves this timely recovery. Third, the duration of migration is shorter than that of live migration in most cases, because of one-time state transfer without retransmission. These characteristics conform to the metrics of importance, as defined in Section 5.1.3.

In addition, enlightened post-copy is resilient to varying workloads in the VM, com-

Table 5.1: Costs of Idle VM Migration. The tables show time and state transfer statistics of migrating an idle VM, with no active applications inside the guest. The numbers in parentheses in part (b) represent a percentage of the total transfer amount.

(Unit: ms)	Enlightenment	Suspension	Execution Transfer	Total Duration
Live	-	6327	6548	6548
EPC	691	706	1280	2571

(a) Time Metrics

(Unit: KB)	Pre-Copy	Demand Fetch	Push	Free Memory Information	Total
Live	1029496 (100%)	-	-	-	1029496
EPC	36028 (4.0%)	892 (0.1%)	871188 (95.9%)	10 (<<0.1%)	908118

(b) State Transfer Amounts

pared to stop-and-copy and simple post-copy. Similar to live migration, these approaches have one particular metric of focus. Their effectiveness, therefore, largely depends on the nature of the VM workload, whereas enlightened post-copy at least has performance comparable to the best approach among them. The cost of achieving such efficiency is insignificant when migrating idle VMs, and moderate in terms of memory management inside the guest OS.

5.5 Summary

In this chapter, we described enlightened post-copy, an approach to urgently migrating VMs under contention. It addresses aspects of VM migration differing from the focus of the existing approaches: urgent execution transfer of the migrated VM and fast aggregate performance recovery of the contending VMs. The current standard of migration, live migration, exhibits undesirable characteristics in these aspects due to its design choices. Departing from its blackbox nature, we treat migration as a native functionality of the guest OS and have it generate current execution knowledge. Enlightened post-copy exploits this

cooperation between the guest OS and the hypervisor, allowing prioritized post-copy state transfer that achieves the above objectives. Our prototype, implemented in guest Linux and qemu-kvm, requires only moderate changes to the guest kernel, and it demonstrates that the cooperative approach resolves the contention between VMs up to several times faster than live migration.

Chapter 6

Conclusion

The executability guarantee of virtualization provides the capability for VMs to be transferred between machines. It allows deploying the software encapsulated in these VMs beyond the constraints of the physical boundaries. However, the sheer size of VM state makes it challenging to use this strong feature effectively, especially under the existence of resource limitations. In this thesis, we proposed the use of execution knowledge in VM state transfer to enhance its efficiency. The resulting efficiency leads to practical solutions that address the timeliness requirements in two concrete contexts: VM delivery over WANs and urgent transfer of VMs under contention.

In Chapter 3, we described our solution to the challenges in VM delivery over WANs, called `vTube`. In this problem domain, the main resource constraint is limited bandwidth between the server storing virtual appliance images and the client that resides across the WAN. Straightforward approaches such as entire or partial VM image downloading can easily take tens of minutes or even hours in such environments. `vTube` exploits past execution knowledge derived from instances of each virtual appliance, in order to instantiate VMs within a time frame of minutes while sustaining good performance of their execution. Specifically, the system adopts the video streaming paradigm and dynamically delivers VM state depending on the behavior of the current VM executing on the client. Its streaming algorithm analyzes memory and disk state accesses in the past execution traces, extracting and accumulating access pattern knowledge specific to each virtual appliance. During user sessions, this knowledge is used to estimate parts of VM state that will be accessed in the near future, provided the current course of VM execution. Our results demonstrated that `vTube` efficiently streams virtual appliance images to the client over real-world networks, including 3G, 4G LTE, and public Wi-Fi's.

In Chapter 4, we investigated aspects of `vTube`'s usability that elude systems-level

evaluation by a series of user studies. The value of these studies is three-fold. First, we studied the impact of the streaming model of vTube on its performance, as perceived by human users. Second, we compared the vTube model to a common alternative approach, remote VM access, with VNC. Third, we evaluated the efficacy of vTube under varied network conditions, especially its effectiveness in absorbing the effect of high round-trip latency expected in WAN environments. We designed and administered the studies in a managed-network setting with applications ranging from document and image editing software to an interactive game. The usability of vTube and VNC was evaluated from multiple perspectives: satisfaction with the system performance, feelings of annoyance, mental demands, and sense of accomplishment. The ratings by the study participants indicated the competitive performance of vTube's VM streaming model in comparison to VNC's remote VM access, over a range of network conditions targeting WANs and wireless networks. In addition, the study results also provided key insights into the impact of latency on user interaction and the implications of using VM streaming as opposed to VM remote access.

In Chapter 5, we discussed enlightened post-copy, a new approach to urgent transfer of VMs under contention. In the context of this problem, the resource limitation of a machine hosting multiple VMs makes migration a time-critical operation for resolving their contention. Existing migration methods do not adequately address this aspect of load balancing that requires urgency; in particular, pre-copy-based live migration often suffers an elongated period of time until the execution of the migrated VM is transferred, meanwhile letting the contention persist. Enlightened post-copy leverages current execution knowledge of the target VM to urgently resolve this contention. The guest OS natively supports migration, and informs the hypervisor about how to prioritize memory state during its transfer. Using this explicit knowledge, the hypervisor immediately suspends the VM on the source and resumes it on the destination, while performing state transfer in a post-copy manner as instructed by the guest OS. Our experiments highlight the comparison between enlightened post-copy and live migration, with highly loaded VMs that run common server applications such as Memcached, MySQL, and Cassandra. The results showed the effectiveness of enlightened post-copy in 1) resolving the contention between VMs immediately, 2) recovering the aggregate throughput of the VMs under contention, and 3) mitigating the performance degradation of the migrated VM through the use of execution knowledge. Specifically, enlightened post-copy recovers the performance of the contending VMs up to several times faster than live migration.

Through the validation steps above, we verified that execution knowledge achieves timely VM state transfer under the existence of resource limitations. vTube employs past execution knowledge that overcomes low bandwidth and high latency while sustain-

ing the system usability, as confirmed by our systems-level and human-centric evaluation. Enlightened post-copy exploits current execution knowledge from the guest OS that allows explicit state prioritization, thereby addressing tight requirements for the time frame of migration. Execution knowledge thus liberates VMs from the physical boundary between machines. While the size of VMs continues to grow today, it reinforces one of the strongest features of virtualization: remote instantiation of computation through preserved executability.

6.1 Future Work

We conceive additional research and experiments with our systems that will further strengthen their value. We first enumerate those directly relevant to VM streaming, comparison of thick and thin clients, and VM migration exploiting enlightenment. Next, we discuss research questions on execution knowledge and encapsulation of computation, inspired by this thesis work.

6.1.1 Enhancements to VM Streaming

We envision extensions to the current `vTube` model in a few notable aspects. First, the users would benefit from a feature to stream updated versions of original virtual appliance images. Their creators may apply updates to the contained applications. Alternatively, the users may save or distribute an image that captures the state after completing some work using the original image. We could potentially apply the access pattern knowledge for the original image to the new versions, avoiding the collection of their execution traces started anew. The access pattern differences between the old and new versions would then be used for streaming the latter. Also, separating the mutable state into a data store would facilitate streaming in these cases. The virtual appliance would include the main executable of an application, and its configuration files and other associated data would reside in cloud storage that is accessible across WANs. Second, smart use of the content-addressable cache would improve the streaming performance. It could be pre-populated with chunk contents that are commonly found in many images, such as those of OS binaries and user libraries. Additionally, when the available storage size for the cache is limited, policies for content eviction would be necessary. Such policies could be based on a basic strategy, for example Least Recently Used (LRU), which removes the contents that have not been accessed for the longest time. Furthermore, they could be augmented with `vTube`'s access pattern knowledge, and could evict those contents that are most unlikely to be accessed

before some time in the future.

6.1.2 Thick Clients and Thin Clients

Thick clients and thin clients have contrasting computing models and factors affecting their usability. These differences could also be considered complementary, and exploited for constructing a hybrid system. One possibility is to have two VM instances, one running on the client and the other on the server, that are periodically synchronized. When the network between the server and client has particularly low bandwidth, the user interface is connected to the server VM using the thin-client model. When the round-trip latency is high, instead, the client VM is used with the thick-client model of VM streaming. Eventually, when the entire VM state has been cached locally, the client VM takes over this hybrid mode of operation.

Another topic worth noting is future network improvements, which impact the relative usability of thick and thin clients. As bandwidth increases, buffering events of `vTube` will have shorter duration given the same VM state size. As round-trip latency lowers, interactive performance of VNC-like systems becomes better. The measurements in Chapters 3 and 4 used bandwidth and latency values that correspond to ranges typical in WAN environments, including 3G, 4G LTE, and public Wi-Fi's. While the bandwidth and round-trip latency in these networks are expected to improve in the future, the latter tends to have more physical constraints; the number of hops depends on the deployed structure of the Internet, and the speed of network signals is unlikely to exceed that of optical fiber cables. Future work would confirm whether these expectations hold, in which case there would be better prospects for thick clients.

6.1.3 Extensions to Enlightened VM Migration

As a migration method, enlightened post-copy serves as a building block for load balancing schemes, and could work with cluster-level management for further effectiveness. For example, before the start of migration, a load balancer could divert service requests for the target VM to other VMs on the same host. The load of the target VM would then be temporarily reduced, and its disruption would have less impact on the aggregate performance of the VMs. Once the migration completes, service requests could be directed to the migrated VM again on its destination. Integration of enlightened post-copy into higher-level resource managers in such a manner would strengthen its benefits, while reducing its performance costs.

Enlightened post-copy could also incorporate additional sources of enlightenment from the application level. Java execution environments, for example, keep track of how memory regions are used for objects. Similar to JAVMM [58], such information could be passed to the guest OS to assist fine-grained categorization of memory pages. When taking this approach, we are adding a level of inter-layer interaction, between the applications and guest OS. In enlightened post-copy, the guest OS supports migration transparently to the applications. Creating the information flow between the applications and guest OS, however, means that the former need modifications. For this reason, a clean interface for passing the relevant information would be an important research question to consider.

The type of enlightenment, as well as the interface, determines its applicability. The categories of memory pages used in enlightened post-copy follow general concepts such as kernel-or-user and code-or-data. These distinctions reflect the memory management by x86 architectures, which has a privilege level and permissions assigned to each page. Therefore, while we implemented the mechanism in Linux, we believe that our use of enlightenment is applicable to Windows and other OSes without significant changes. Investigating the practicality of implementation for other platforms would validate this generality of our approach.

6.1.4 Source of Execution Knowledge

Execution knowledge can be formed in various ways. Except for the acquisition of free memory lists, `vTube` uses a blackbox approach, which does not require any modifications to the guest environment. Enlightened post-copy, on the other hand, takes a whitebox approach with mechanisms implemented inside the guest OS. These design choices in obtaining execution knowledge are driven by their target contexts. In `vTube`, past execution knowledge proves to be effective in estimating the state accesses of current VM execution. This knowledge can be obtained without explicit guest cooperation, and accumulated for future use. In enlightened post-copy, up-to-date execution knowledge needs to be immediately accessible and applicable. Enlightenment, for this reason, is a suitable candidate as the source of execution knowledge. The blackbox and whitebox approaches thus make trade-offs between the time frame of applying the knowledge and the cost of implementation. In addition to these options, graybox techniques [25] offer solutions in the middle. They leverage knowledge that can be inferred about the target entity without modifying it, such as behavior expected from internal algorithms. The selection of what approach to use depends on the attributes desired in the right solution. Hybrid approaches are also conceivable. In the context of migration, for example, periodic monitoring of the VM workload could judge whether live migration would provide sufficient timeliness; other-

wise, if the guest OS supports enlightened post-copy, it could be triggered to better satisfy the performance requirements. Investigation into the proper selection, or combination, of these different solutions would be a useful extension to our work.

6.1.5 Future Abstraction for Encapsulating Computation

Finally, this thesis motivates further exploration of virtualization as the unit of encapsulating computation. Virtualization is the lowest-level mechanism for encapsulation, targeting that of the entire software stack and requiring only the hypervisor for its executability. With `vTube` and enlightened post-copy, we demonstrated that this holistic, and seemingly heavy, approach can achieve movability between machines through sophistication in state transfer. Alternatives at higher levels can provide efficiency, with appropriate support needed above the layer of the hypervisor. OS-level mechanisms, such as Linux containers and FreeBSD jails, only need to move the state of the target process. The OSes involved, however, must be kept compatible, which likely is a more cumbersome task than having hypervisors consistent across machines. Application-level solutions, as exemplified by the binary compatibility of Java, have further dependencies including the run-time environment. An important research question would be what the most promising abstraction for computation is in the coming era. OS-level and process-level techniques may be increasingly supported by common platforms. Still, virtualization remains to be a core technology that offers robust executability of the guest environments and strong isolation between them. Also, inter-layer approaches to efficiency, as used in enlightened post-copy, and the abstraction for their information flow would be factors that determine the efficacy of these various options.

6.2 Concluding Remarks

Timely transfer of VMs augments their capability to preserve executability, expanding its applicability to diverse contexts. In this thesis, we proposed the use of execution knowledge to achieve this timeliness. Our work has shown that the robust execution environment for applications and its movability between physical machines are not mutually exclusive; virtualization remains to be an attractive solution with both of these features, among various approaches to decoupling software from hardware. We envision that the efficiency achieved through execution knowledge will foster the exploration of its application in different forms, and impact the ways of abstracting computational units for remote instantiation.

Bibliography

[1] Apposite Technologies :: Linktropy 5500 WAN Emulator. http://www.apposite-tech.com/products/5500.html.

[2] Arcanum product page (archived). http://www.gog.com/game/arcanum_of_steamworks_and_magick_obscura.

[3] Avidemux - Main Page. http://fixounet.free.fr/avidemux.

[4] AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting. http://aws.amazon.com/ec2.

[5] FUSE: Filesystem in Userspace. http://fuse.sourceforge.net.

[6] Heroes of Might and Magic IV. http://www.gog.com/game/heroes_of_might_and_magic_4_complete.

[7] KVM. http://www.linux-kvm.org/page/Main_Page.

[8] memcached - a distributed memory object caching system. http://memcached.org.

[9] Microsoft Word | Document and Word Processing Software. https://products.office.com/en-us/word.

[10] MPlayer - The Movie Player.

[11] NASA TLX Homepage. http://humansystems.arc.nasa.gov/groups/tlx/.

[12] OLTPBenchmark. http://oltpbenchmark.com/wiki.

[13] Oracle VM VirtualBox. https://www.virtualbox.org.

[14] Photoshop | Photoshop.com. http://www.photoshop.com/products/photoshop.

[15] PopCap Games | Bejeweled Twist - Free Online Games. http://www.popcap.com/games/bejeweled-twist/online.

[16] Riven: the sequel to Myst. http://www.gog.com/gamecard/riven_the_sequel_to_myst.

[17] Selenium - Web Browser Automation. http://docs.seleniumhq.org.

[18] State of the Internet Report | Akamai. http://www.akamai.com/stateoftheinternet.

[19] The Apache Cassandra Project. http://cassandra.apache.org.

[20] userfaultfd v4 [LWN.net]. https://lwn.net/Articles/644532.

[21] Virtio - KVM. http://www.linux-kvm.org/page/Virtio.

[22] Yahoo! Cloud Serving Benchmark (YCSB). https://github.com/brianfrankcooper/YCSB/wiki.

[23] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper. Predicting the Performance of Virtual Machine Migration. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*, Miami Beach, FL, USA, 2010. IEEE.

[24] Samer Al-Kiswany, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. VMFlock: Virtual Machine Co-migration for the Cloud. In *Proceedings of the Twentieth International Symposium on High Performance Distributed Computing (HPDC '11)*, San Jose, CA, USA, 2011. ACM.

[25] Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau. Information and Control in Gray-box Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*, Banff, AB, Canada, 2001. ACM.

[26] Amnon Barak and Oren La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Future Generation Computer Systems — Special Issue on HPCN '97*, 13(4-5), March 1998.

[27] Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh. THINC: A Virtual Display Architecture for Thin-client Computing. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05)*, Brighton, UK, 2005. ACM.

[28] Ricardo A. Baratto, Shaya Potter, Gong Su, and Jason Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the Tenth Annual International Conference on Mobile Computing and Networking (MobiCom '04)*, Philadelphia, PA, USA, 2004. ACM.

[29] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, USA, 2003. ACM.

[30] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *Proceedings of the Third International Conference on Virtual Execution Environments (VEE '07)*, San Diego, CA, USA, 2007. ACM.

[31] David Breitgand, Gilad Kutiel, and Danny Raz. Cost-aware Live Migration of Services in the Cloud. In *Proceedings of the Eleventh USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '11)*, Boston, MA, USA, 2011. USENIX Association.

[32] Angela Demke Brown, Todd C. Mowry, and Orran Krieger. Compiler-based I/O Prefetching for Out-of-Core Applications. *ACM Transactions on Computer Systems*, 19(2), May 2001.

[33] Roy Bryant, Alexey Tumanov, Olga Irzak, Adin Scannell, Kaustubh Joshi, Matti Hiltunen, Andrés Lagar-Cavilla, and Eyal de Lara. Kaleidoscope: Cloud Micro-elasticity via VM State Coloring. In *Proceedings of the Sixth ACM European Conference on Computer Systems (EuroSys '11)*, Salzburg, Austria, April 2011. ACM.

[34] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems*, 15(4), November 1997.

[35] T. W. Butler. Computer Response Time and User Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '83)*, Boston, MA, USA, 1983. ACM.

[36] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A Study of Integrated Prefetching and Caching Strategies. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '95/PERFORMANCE '95)*, Ottawa, ON, Canada, 1995. ACM.

[37] Pei Cao, Edward W. Felten, and Kai Li. Implementation and Performance of Application-controlled File Caching. In *Proceedings of the First USENIX Conference on Operating Systems Design and Implementation (OSDI '94)*, Monterey, CA, USA, 1994. USENIX Association.

[38] Ramesh Chandra, Nickolai Zeldovich, Constantine Sapuntzakis, and Monica S. Lam. The Collective: A Cache-based System Management Architecture. In *Proceedings of the Second Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI '05)*, Boston, MA, USA, 2005. USENIX Association.

[39] Fay Chang and Garth A. Gibson. Automatic I/O Hint Generation Through Speculative Execution. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, New Orleans, LA, USA, 1999. USENIX Association.

[40] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Proceedings of the Second Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI '05)*, Boston, MA, USA, 2005. USENIX Association.

[41] R. J. Creasy. The Origin of the VM/370 Time-sharing System. *IBM Journal of Research and Development*, 25(5), September 1981.

[42] James R. Dabrowski and Ethan V. Munson. Is 100 Milliseconds Too Fast? In *CHI '01 Extended Abstracts on Human Factors in Computing Systems (CHI EA '01)*, Seattle, WA, USA, 2001. ACM.

[43] Gary L. Dannenbring. The effect of computer response time on user performance and satisfaction: A preliminary investigation. *Behavior Research Methods & Instrumentation*, 15(2), March 1983.

[44] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. ElasTraS: An Elastic, Scalable, and Self-managing Transactional Database for the Cloud. *ACM Transactions on Database Systems*, 38(1), April 2013.

[45] Umesh Deshpande, Yang You, Danny Chan, Nilton Bila, and Kartik Gopalan. Fast Server Deprovisioning Through Scatter-Gather Live Migration of Virtual Machines. In *Proceedings of the 2014 IEEE International Conference on Cloud Computing (CLOUD '14)*, Anchorage, AK, USA, 2014. IEEE.

[46] Fred Douglis and John Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software — Practice & Experience*, 21(8), July 1991.

[47] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*, Athens, Greece, 2011. ACM.

[48] T. J. Goodman and Robert Spence. The Effect of Computer System Response Time Variability on Interactive Graphical Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(3), March 1981.

[49] Kinshuk Govil, Dan Teodosiu, Yongqiang Huang, and Mendel Rosenblum. Cellular Disco: Resource Management Using Virtual Clusters on Shared-memory Multiprocessors. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[50] James Griffioen and Randy Appleton. Reducing File System Latency Using a Predictive Approach. In *Proceedings of the USENIX Summer 1994 Technical Conference*, Boston, MA, USA, 1994. USENIX Association.

[51] Mitchell Grossberg, Raymond A. Wiesen, and Douwe B. Yntema. Experiment on Problem Solving with Delayed Computer Responses. In *IEEE Transactions on Systems, Man and Cybernetics*, volume SMC-6, March 1976.

[52] Tian Guo, Vijay Gopalakrishnan, K. K. Ramakrishnan, Prashant Shenoy, Arun Venkataramani, and Seungjoon Lee. VMShadow: Optimizing the Performance of Latency-sensitive Virtual Desktops in Distributed Clouds. In *Proceedings of the Fifth ACM Multimedia Systems Conference (MMSys '14)*, Singapore, Singapore, 2014. ACM.

[53] Jan L. Guynes. Impact of System Response Time on State Anxiety. *Communications of the ACM*, 31(3), March 1988.

[54] Stuart Hacking and Benoît Hudzia. Improving the Live Migration Process of Large Enterprise Applications. In *Proceedings of the Third International Workshop on Virtualization Technologies in Distributed Computing (VTDC '09)*, Barcelona, Spain, 2009. ACM.

[55] Jacob Gorm Hansen and Eric Jul. Self-migration of Operating Systems. In *Proceedings of the Eleventh Workshop on ACM SIGOPS European Workshop*, Leuven, Belgium, 2004. ACM.

[56] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy Live Migration of Virtual Machines. *SIGOPS Operating Systems Review*, 43(3), July 2009.

[57] Michael R. Hines and Kartik Gopalan. Post-copy Based Live Virtual Machine Migration Using Adaptive Pre-paging and Dynamic Self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*, Washington, DC, USA, 2009. ACM.

[58] Kai-Yuan Hou, Kang G. Shin, and Jan-Lung Sung. Application-assisted Live Migration of Virtual Machines with Java Applications. In *Proceedings of the Tenth ACM European Conference on Computer Systems (EuroSys '15)*, Bordeaux, France, 2015. ACM.

[59] John A. Hoxmeier and C DiCesare. System Response Time and User Satisfaction : An Experimental Study of Browser-based Applications. *Proceedings of the Association of Information Systems Americas Conference*, 2000.

[60] Wenjin Hu, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, and Jeanna N. Matthews. A Quantitative Study of Virtual Machine Live Migration. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC '13)*, Miami, FL, USA, 2013. ACM.

[61] Binbin Huang, Rongheng Lin, Kai Peng, Hua Zou, and Fangchun Yang. Minimizing Latency in Fetching Virtual Machine Images Based on Multi-point Collaborative Approach. In *Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (GREENCOM-ITHINGS-CPSCOM '13)*, Beijing, China, 2013. IEEE.

[62] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High Performance Virtual Machine Migration with RDMA over Modern Interconnects. In *Proceedings*

of the 2007 IEEE International Conference on Cluster Computing (CLUSTER '07), Austin, TX, USA, 2007. IEEE.

[63] Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. Optimized Pre-copy Live Migration for Memory Intensive Applications. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, Seattle, WA, USA, 2011. ACM.

[64] Changyeon Jo and Bernhard Egger. Optimizing Live Migration for Virtual Desktop Clouds. In *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science - Volume 01 (CloudCom '13)*, Bristol, UK, 2013. IEEE.

[65] Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient Live Migration of Virtual Machines Using Shared Storage. In *Proceedings of the Ninth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13)*, Houston, TX, USA, 2013. ACM.

[66] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. How Fast is Fast Enough?: A Study of the Effects of Latency in Direct-touch Pointing Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*, Paris, France, 2013. ACM.

[67] Poul-Henning Kamp and Robert N. M. Watson. Jails: Confining the omnipotent root. In *Proceedings of the Second International SANE Conference*, Maastricht, Netherlands, 2000. Stichting SANE.

[68] Joeng Kim, Ricardo A. Baratto, and Jason Nieh. pTHINC: A Thin-client Architecture for Mobile Wireless Web. In *Proceedings of the Fifteenth International Conference on World Wide Web (WWW '06)*, Edinburgh, Scotland, 2006. ACM.

[69] Tracy Kimbrel, Andrew Tomkins, R. Hugo Patterson, Brian Bershad, Pei Cao, Edward W. Felten, Garth A. Gibson, Anna R. Karlin, and Kai Li. A Trace-driven Comparison of Algorithms for Parallel Prefetching and Caching. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation (OSDI '96)*, Seattle, WA, USA, 1996. ACM.

[70] Akane Koto, Hiroshi Yamada, Kei Ohmura, and Kenji Kono. Towards Unobtrusive VM Live Migration for Cloud Computing Platforms. In *Proceedings of the Third ACM SIGOPS Asia-Pacific Conference on Systems (APSys '12)*, Seoul, South Korea, 2012. USENIX Association.

[71] Michael A. Kozuch and Mahadev Satyanarayanan. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, USA, 2002. IEEE.

[72] W Kuhmann. Experimental investigation of stress-inducing properties of system response times. *Ergonomics*, 32(3), March 1989.

[73] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and Mahadev Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proceedings of the Fourth ACM European Conference on Computer Systems (EuroSys '09)*, Nuremberg, Germany, 2009. ACM.

[74] Albert Lai and Jason Nieh. Limits of Wide-area Thin-client Computing. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '02)*, Marina Del Rey, CA, USA, 2002. ACM.

[75] Frederic Lardinois. Akamai: Average U.S. Internet Speed Up 28% YoY, Now At 7.4 Mbps, But South Korea, Japan And Hong Kong Still Far Ahead. *TechCrunch*, April 2013.

[76] Zhenmin Li, Zhifeng Chen, and Yuanyuan Zhou. Mining Block Correlations to Improve Storage Performance. *ACM Transactions on Storage*, 1(2), May 2005.

[77] Jochen Liedtke. On Micro-kernel Construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP '95)*, Copper Mountain, CO, USA, 1995. ACM.

[78] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In *Proceedings of the Eighteenth ACM International Symposium on High Performance Distributed Computing (HPDC '09)*, Garching, Germany, 2009. ACM.

[79] James Lo, Eric Wohlstadter, and Ali Mesbah. Live Migration of JavaScript Web Apps. In *Proceedings of the Twenty-Second International Conference on World Wide Web (WWW '13 Companion)*, Rio de Janeiro, Brazil, 2013. International World Wide Web Conferences Steering Committee.

[80] Peng Lu, Antonio Barbalace, and Binoy Ravindran. HSG-LM: Hybrid-copy Speculative Guest OS Live Migration Without Hypervisor. In *Proceedings of the Sixth*

International Systems and Storage Conference (SYSTOR '13), Haifa, Israel, 2013. ACM.

[81] Anil Madhavapeddy, Thomas Leonard, Magnus Skjegstad, Thomas Gazagnaire, David Sheets, Dave Scott, Richard Mortier, Amir Chaudhry, Balraj Singh, Jon Ludlam, Jon Crowcroft, and Ian Leslie. Jitsu: Just-in-time Summoning of Unikernels. In *Proceedings of the Twelfth USENIX Conference on Networked Systems Design and Implementation (NSDI '15)*, Oakland, CA, USA, 2015. USENIX Association.

[82] Anil Madhavapeddy and David J. Scott. Unikernels: The Rise of the Virtual Library Operating System. *Communications of the ACM*, 57(1), January 2014.

[83] Ali José Mashtizadeh, Min Cai, Gabriel Tarasuk-Levin, Ricardo Koller, Tal Garfinkel, and Sreekanth Setty. XvMotion: Unified Virtual Machine Migration over Long Distance. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC '14)*, Philadelphia, PA, USA, 2014. USENIX Association.

[84] Robert B. Miller. Response Time in Man-computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS '68 (Fall, part I))*, San Francisco, CA, USA, 1968. ACM.

[85] Grzegorz Miłós, Derek G. Murray, Steven Hand, and Michael A. Fetterman. Satori: Enlightened Page Sharing. In *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX ATC '09)*, San Diego, CA, USA, 2009. USENIX Association.

[86] Takeshi Mishima and Yasuhiro Fujiwara. Madeus: Database Live Migration Middleware Under Heavy Workloads for Cloud Environment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, Melbourne, Australia, 2015. ACM.

[87] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic Compiler-inserted I/O Prefetching for Out-of-Core Applications. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation (OSDI '96)*, Seattle, WA, USA, 1996. ACM.

[88] Fiona F. Nah. A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, 23(3), May 2004.

[89] Senthil Nathan, Umesh Bellur, and Purushottam Kulkarni. Towards a Comprehensive Performance Model of Virtual Machine Live Migration. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*, Kohala Coast, HI, USA, 2015. ACM.

[90] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast Transparent Migration for Virtual Machines. In *Proceedings of the 2005 USENIX Annual Technical Conference (USENIX ATC '05)*, Anaheim, CA, USA, 2005. USENIX Association.

[91] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. Designing for Low-latency Direct-touch Input. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on User Interface Software and Technology (UIST '12)*, Cambridge, MA, USA, 2012. ACM.

[92] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the Fifth Symposium on Operating Systems Design and implementation (OSDI '02)*, Boston, MA, USA, 2002. ACM.

[93] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP '95)*, Copper Mountain, CO, USA, 1995. ACM.

[94] Chunyi Peng, Minkyong Kim, Zhe Zhang, and Hui Lei. VDN: Virtual Machine Image Distribution Network for Cloud Data Centers. In *Proceedings of INFOCOM 2012*, Orlando, FL, USA, 2012. IEEE.

[95] Michael L. Powell and Barton P. Miller. Process Migration in DEMOS/MP. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles (SOSP '83)*, Bretton Woods, NH, USA, 1983. ACM.

[96] Joshua Reich, Oren Laadan, Eli Brosh, Alex Sherman, Vishal Misra, Jason Nieh, and Dan Rubenstein. VMTorrent: Virtual Appliances On-demand. In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*, New Delhi, India, 2010. ACM.

[97] Joshua Reich, Oren Laadan, Eli Brosh, Alex Sherman, Vishal Misra, Jason Nieh, and Dan Rubenstein. VMTorrent: Scalable P2P Virtual Machine Streaming. In *Proceedings of the Eighth International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, Nice, France, 2012. ACM.

[98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), January 1998.

[99] Pierre Riteau, Christine Morin, and Thierry Priol. Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and

Content-Based Addressing. In *Proceedings of the Seventeenth International European Conference on Parallel and Distributed Computing (Euro-Par '11)*, Bordeaux, France, August 2011. Springer-Verlag.

[100] Avi Rushinek and Sara F. Rushinek. What Makes Users Happy? *Communications of the ACM*, 29(7), July 1986.

[101] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual Appliances for Deploying and Maintaining Software. In *Proceedings of the Seventeenth USENIX Conference on System Administration (LISA '03)*, San Diego, CA, USA, 2003. USENIX Association.

[102] Constantine Sapuntzakis and Monica S. Lam. Virtual Appliances in the Collective: A Road to Hassle-free Computing. In *Proceedings of the Ninth Conference on Hot Topics in Operating Systems - Volume 9 (HotOS '03)*, Lihue, HI, USA, 2003. USENIX Association.

[103] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, 2002. ACM.

[104] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), October 2009.

[105] Mahadev Satyanarayanan, Vasanth Bala, Gloriana St. Clair, and Erika Linke. Collaborating with Executable Content Across Space and Time. In *Proceedings of the Seventh International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2011)*, Orlando, FL, USA, 2011. IEEE.

[106] Mahadev Satyanarayanan, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, David J. Farber, Michael A. Kozuch, Casey J. Helfrich, Partho Nath, and H. Andrés Lagar-Cavilla. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing*, 11(2), March 2007.

[107] Stephen Shankland. VMware opens virtual-appliance marketplace. *CNET News*, November 2006.

[108] Aidan Shribman and Benoit Hudzia. Pre-Copy and Post-copy VM Live Migration for Memory Intensive Applications. In *Proceedings of the Eighteenth International Conference on Parallel Processing Workshops (Euro-Par '12)*, Rhodes Island, Greece, 2013. Springer-Verlag.

[109] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In *Proceedings of the Second ACM European Conference on Computer Systems (EuroSys '07)*, Lisbon, Portugal, 2007. ACM.

[110] Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. Parallelizing Live Migration of Virtual Machines. In *Proceedings of the Ninth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '13)*, Houston, TX, USA, 2013. ACM.

[111] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In *Proceedings of the Seventh ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, USA, 2011. ACM.

[112] Petter Svärd, Benoit Hudzia, Steve Walsh, Johan Tordsson, and Erik Elmroth. Principles and Performance Characteristics of Algorithms for Live VM Migration. *SIGOPS Operating Systems Review*, 49(1), January 2015.

[113] Niraj Tolia, David G. Andersen, and Mahadev Satyanarayanan. Quantifying Interactive User Experience on Thin Clients. *Computer*, 39(3), March 2006.

[114] Andrew Tomkins, R. Hugo Patterson, and Garth Gibson. Informed Multi-process Prefetching and Caching. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '97)*, Seattle, WA, USA, 1997. ACM.

[115] Carl A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, 2002. ACM.

[116] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive Process-level Live Migration and Back Migration in HPC Environments. *Journal of Parallel and Distributed Computing*, 72(2), February 2012.

[117] Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Grible. Constructing Services with Interposable Virtual Hardware. In *Proceedings of the First Conference on Symposium on Networked Systems Design and Implementation - Volume 1 (NSDI '04)*, San Francisco, CA, USA, 2004. USENIX Association.

[118] Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus van der Merwe. CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *Proceedings of the Seventh ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, USA, 2011. ACM.

[119] Irene Zhang, Alex Garthwaite, Yury Baskakov, and Kenneth C. Barr. Fast Restore of Checkpointed Memory Using Working Set Estimation. In *Proceedings of the Seventh ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, Newport Beach, CA, USA, 2011. ACM.